



HAL
open science

Quantum variational optimization methods and their applications

Yagnik Chatterjee

► **To cite this version:**

Yagnik Chatterjee. Quantum variational optimization methods and their applications. Micro and nanotechnologies/Microelectronics. Université de Montpellier, 2024. English. NNT : 2024UMONS022 . tel-04766221

HAL Id: tel-04766221

<https://theses.hal.science/tel-04766221v1>

Submitted on 4 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale - Information Structures Systèmes (ED166)

Unité de recherche - Laboratoire d'informatique, de robotique et de microélectronique de
Montpellier (LIRMM), UMR 5506

Méthodes d'optimisation variationnelles quantiques et leurs applications Quantum variational optimization methods and their applications

Présentée par Yagnik CHATTERJEE
le 9 octobre 2024

Sous la direction de Éric BOURREAU

Devant le jury composé de

Gilles TROMBETTONI, Professeur, Université de Montpellier
Aida TODRI-SANIAL, Full Professor, Eindhoven University of Technology
Philippe LACOMME, Professeur, Université Clermont Auvergne
Margarita VESHCHEROVA, Docteure, Terra Quantum
Éric BOURREAU, Maître de Conférences, Université de Montpellier
Marko RANČIĆ, Professor (W1), Heidelberg University
Wesley COELHO, Docteur, PASQAL
Jean-Patrick MASCOMERE, Scientific Computing Manager, TotalEnergies

Président du jury
Rapporteuse
Rapporteur
Examinatrice
Directeur de thèse
Co-encadrant
Invité
Invité



UNIVERSITÉ
DE MONTPELLIER

Abstract

Quantum computing is a rapidly developing field that has seen a huge amount of interest in the last couple of decades due to its promise of revolutionizing several domains of business and science. It presents a new way of doing computations by making use of fundamental properties of quantum mechanics such as superposition and entanglement. Combinatorial optimization, on the other hand, is a field that is omnipresent in the industry and where small improvements can have a significant impact. This thesis aims to tackle optimization problems using quantum algorithms.

NP-hard optimization problems are not believed to be exactly solvable through general polynomial time algorithms. Variational quantum algorithms (VQAs) to address such combinatorial problems have been of great interest recently. Such algorithms are heuristic and aim to obtain an approximate solution. The hardware, however, is still in its infancy and the current Noisy Intermediate Scale Quantum (NISQ) computers are not able to optimize industrially relevant problems. An issue with contemporary quantum optimization algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) is that they scale linearly with problem size. To tackle this issue, we present the LogQ encoding, using which we can design quantum variational algorithms that scale logarithmically with problem size – opening an avenue for treating optimization problems of unprecedented scale on gate-based quantum computers. We show how this algorithm can be applied to several combinatorial optimization problems such as Maximum Cut, Minimum Partition, Maximum Clique and Maximum Weighted Independent Set (MWIS). Subsequently, these algorithms are tested on a quantum simulator with graph sizes of over a hundred nodes and on a real quantum computer up to graph sizes of 256. To our knowledge, these constitute the largest realistic combinatorial optimization problems ever run on a NISQ device, overcoming previous problem sizes by almost tenfold.

Next, we apply the LogQ encoding to two use-cases for large companies such as TotalEnergies. Fleet conversion is the process of transitioning a fleet of vehicles to more sustainable and environmentally friendly alternatives. It is modeled as a column generation scheme with the MWIS problem as the sub-problem or worker problem. We use the LogQ method to solve the MWIS Workers and demonstrate how quantum and classical solvers can be used together in a hybrid manner to approach an industrial-sized use-case.

Mesh segmentation refers to the process of dividing a complex mesh (composed of vertices, edges, and faces) into meaningful and semantically coherent parts or regions. Mesh segmentation plays an important part in computer modeling, which is extensively used in the core domains of TotalEnergies' activities such as Earth imaging, physical modeling for reservoirs, and others. We define the problem as a graph optimization problem and use the LogQ encoding to solve it.

Résumé

L'informatique quantique est un domaine en plein essor qui a suscité un intérêt considérable au cours des deux dernières décennies en raison de sa promesse de révolutionner plusieurs domaines. Il s'agit d'une nouvelle façon d'effectuer des calculs en utilisant les propriétés fondamentales de la mécanique quantique telles que la superposition et l'intrication. La recherche opérationnelle et plus particulièrement l'optimisation combinatoire, quant à elle, est un domaine omniprésent dans l'industrie où de petites améliorations peuvent avoir un impact significatif. Cette thèse vise à résoudre des problèmes d'optimisation combinatoire à l'aide d'algorithmes quantiques.

Les problèmes d'optimisation NP-difficiles ne sont pas considérés comme pouvant être résolus exactement par des algorithmes généraux en temps polynomial. Les algorithmes quantiques variationnels (VQA en anglais) destinés à attaquer ces problèmes combinatoires ont récemment suscité une grande attention. Ces algorithmes sont heuristiques et visent à obtenir une solution approchée. Cependant, le matériel n'en est encore qu'à ses débuts et les ordinateurs quantiques bruités de taille intermédiaire (NISQ en anglais) ne sont pas en mesure d'optimiser les problèmes industriels. Les algorithmes d'optimisation quantique contemporains, tel que le algorithme d'optimisation approchée quantique, Quantum Approximate Optimization Algorithm (QAOA) en anglais, posent un problème : leur échelle est linéaire en fonction de la taille du problème. Pour dépasser cette limite, nous présentons l'encodage LogQ, qui permet de concevoir des algorithmes variationnels quantiques dont l'échelle est logarithmique avec la taille du problème, ce qui ouvre la voie au traitement de problèmes d'optimisation d'une ampleur sans précédent sur des ordinateurs quantiques à portes. Nous montrons comment cet algorithme peut être appliqué sur plusieurs problèmes d'optimisation combinatoire tels que Coupe Maximum, Partition Minimale, Clique Maximale et Stable Maximum Pondéré (MWIS en anglais). Ces algorithmes sont testés sur un simulateur quantique avec des graphes de plus d'une centaine de nœuds et sur un véritable ordinateur quantique jusqu'à des graphes de taille 256. À notre connaissance, il s'agit des plus grands problèmes réalistes d'optimisation combinatoire jamais exécutés sur une machine NISQ à portes, dépassant souvent de près de dix fois la taille des problèmes résolus précédemment.

Ensuite, nous appliquons le codage LogQ à deux cas d'utilisation pour de grandes entreprises telles que TotalEnergies. Premier cas, la conversion de flotte de véhicules est le processus de transition d'une flotte de véhicules vers des alternatives plus durables et plus respectueuses de l'environnement. Il est modélisé comme un schéma de génération de colonnes avec le problème MWIS comme sous-problème. Nous utilisons la méthode LogQ pour résoudre le MWIS et démontrons comment les solveurs quantiques et classiques peuvent être utilisés ensemble pour aborder un cas d'utilisation de taille industrielle. Deuxième cas, la segmentation de maillage fait référence au processus de division d'un maillage complexe (composé de sommets, d'arêtes et de faces) en parties ou régions significatives et sémantiquement cohérentes. La segmentation du maillage joue un rôle important dans la modélisation informatique, qui est largement utilisée dans les domaines clés des activités de TotalEnergies, tels que l'imagerie terrestre ou la modélisation physique des réservoirs. Nous définissons le problème comme un problème d'optimisation de graphe et utilisons l'encodage LogQ pour le résoudre.

Acknowledgments

The journey towards completing this PhD has been an incredible one, and I am grateful to all those who have played a significant role in my success.

First and foremost, I would like to express my deepest gratitude to my thesis director, Éric. His exceptional guidance and unwavering support were instrumental in bringing this thesis to fruition. Our relationship transcended the typical dynamic between professor and PhD student, evolving into one that felt more like colleagues. Despite coming from very different technical backgrounds, we learnt a great deal from each other. Beyond the professional collaboration, I deeply value the camaraderie we shared, which made my visits to Montpellier and my overall experience as a PhD student truly enjoyable.

I would like to extend my gratitude to Marko for his outstanding work, which laid the groundwork for this thesis. His ability to define the project so clearly helped me maintain focus throughout the process. I also gained valuable insights from our numerous discussions, both scientific and otherwise.

I would like to thank Aida and Philippe for taking the time to read my thesis and writing reports on it. I would also like to thank Gilles and Margarita, who very kindly accepted to be in my jury. I thank Wesley for accepting to be in the jury as a guest.

I would like to thank Jean-Patrick for his efficient management of the depleted quantum computing team at TotalEnergies during the final year of my PhD. His efforts to navigate the challenges and lead the team to the best of his ability are greatly appreciated. I also thank him for accepting to be in the jury as a guest.

I would like to extend my gratitude to all my colleagues at TotalEnergies for their support and collaboration over the past three years. In particular, I want to thank Elie, Ali, Nouha, Baptiste, and Benjamin for not only being great collaborators but also wonderful friends.

Finally, I would like to thank my parents, whose unwavering support has been essential in getting me to where I am today—both literally and figuratively.

Introduction		7
1 Background		11
1.1 Elements of quantum computing		11
1.1.1 Qubit		11
1.1.2 Multi-qubit systems		12
1.1.3 Some important properties of Dirac algebra		14
1.1.4 Quantum Gates and Circuits		15
1.1.5 Measurement, Observables and Expectation Value		17
1.1.6 Quantum Hardware		21
1.2 Combinatorial Optimization		23
1.2.1 NP-Hard Problems		25
1.2.2 Integer Linear Programming		26
1.2.3 Approximation Algorithms		28
1.2.4 Quadratic Unconstrained Binary Optimization (QUBO)		28
1.2.5 Goemans Williamson Algorithm		29
1.3 Variational Quantum Algorithms for Optimization		30
1.3.1 Black-box optimizers		32
1.3.2 Quantum Approximate Optimization Algorithm		33
2 The LogQ Encoding: Solving NP-Hard problems using exponentially fewer qubits		37
2.1 Introduction		37
2.2 Methods		38
2.2.1 A qubit-efficient MAXIMUM CUT Algorithm		38
2.2.1.1 Description of the algorithm		38
2.2.1.2 Step-by-step explanation of LogQ for MAXIMUM CUT with an example		41
2.2.1.3 Complexity analysis of LogQ		50
2.2.2 Applying the algorithm to other NP-hard problems		51
2.2.2.1 MINIMUM PARTITION to MAXIMUM CUT		52
2.2.2.2 MAXIMUM 2-SAT to MAXIMUM CUT		53
2.2.2.3 MAXIMUM CLIQUE to MAXIMUM 2-SAT		54
2.2.3 Generalizing the LogQ Encoding		55
2.2.3.1 MAXIMUM WEIGHTED INDEPENDENT SET using QUBO		58

2.2.4	Classical Models for Benchmarking	58
2.2.4.1	Integer Linear Program for MAXIMUM CUT Problem	58
2.2.4.2	Integer Quadratic Program for MINIMUM PARTITION Problem	58
2.3	Computational Results and Discussions	59
2.3.1	MAXIMUM CUT	59
2.3.2	MINIMUM PARTITION as a conversion from MAXIMUM CUT	63
2.3.3	MAXIMUM CLIQUE as a conversion from MAXIMUM CUT	64
2.3.4	MAXIMUM WEIGHTED INDEPENDENT SET using QUBO method	64
2.3.5	A comparative study of time taken by the simulator and the QPU	65
2.4	Conclusion	66
3	A Hybrid Quantum-assisted Column Generation Algorithm for Fleet Conversion	69
3.1	Introduction	69
3.2	Problem Statement and Methods	70
3.2.1	Statement	70
3.2.2	Formal description as a Graph Problem	72
3.2.3	The Column Generation Algorithm	74
3.2.4	Quantum model for the MWIS problem	76
3.2.5	Quantum-assisted algorithm to solve the Coordinator Problem	80
3.3	Experimental Results	81
3.4	Extension: Replacing CWS with better performing QWS	83
3.5	Discussion and Conclusion	84
4	Mesh Segmentation using the LogQ encoding	91
4.1	Background and Context of the Project	91
4.2	Motivation	92
4.3	Mesh Segmentation	93
4.4	Mesh Segmentation as a graph optimization problem	94
4.5	Description of the Objective Function	94
4.6	Modularity for more than two segments	97
4.7	Metrics to evaluate the quality of the segments	98
4.7.1	Rand Index	98
4.7.2	Cut Discrepancy	99
4.8	Results	99
4.9	Conclusion	102
	Conclusion	103

This thesis has been carried out under the joint supervision of TotalEnergies and LIRMM, CNRS - Université de Montpellier. Its primary goal is to explore the field of quantum variational algorithms for combinatorial optimization problems and find interesting industrial use cases for the same. Quantum computing is a rapidly developing field that has seen a huge amount of interest in the last couple of decades due to its promise of revolutionizing several domains of business and science. Optimization, on the other hand, is a field that is omnipresent in the industry and where small improvements can have a significant impact. This thesis aims to tackle optimization problems using quantum computing, straddling both fields in equal measure.

Quantum Computing

Quantum physics has been a part of our daily lives for a long time. The so-called *first quantum revolution* gave us devices such as the transistors, photovoltaic cells and lasers. Today, we cannot imagine our lives without these technologies which have wide-ranging applications such as medical imaging (MRI), TV displays (LEDs) and pretty much everything in our digital lives (PCs, smartphones etc.). During the first quantum revolution we learned to manipulate groups of quantum particles and exploit the interaction between light and matter [1]. Currently we are said to be in the *second quantum revolution* where we manipulate individual quantum objects such as the states of individual particles. The second quantum revolution covers various applications such as quantum computing [2]–[4], quantum communication [5], quantum cryptography [6], [7] and quantum metrology [8]. The scope of this thesis is limited to quantum computing.

Quantum computing is a potentially disruptive field that could have applications in several domains including financial modeling [9], [10], chemistry [11], [12] and optimization [13], [14]. It is a field in which we try to perform numerical computations by exploiting the quantum properties of nature. The unique properties of quantum states such as *superposition* and *entanglement* require us to think about computation in a radically new manner.

While the seeds of quantum computing were sown during the 1970s and 1980s [15]–[17], it is not until recently that there has been a significant advance in terms of quantum hardware, which in turn has led to an immense growth in research on quantum algorithms. We can now run quantum algorithms using high-level programming languages such as Python on real quantum computers (albeit small and noisy) on cloud-based services. Access to quantum hardware, however, remains expensive in terms of both time and money. This is why we need to make sure that the code runs without any error before we run them on a real quantum computer. Consequently, quantum simulators or emulators have become an important part of quantum algorithm research. Quantum simulators are classical comput-

ers or supercomputers that produce exactly the same results a noiseless quantum computer would produce, aiding in the testing process. Moreover, state-of-the-art simulators like the Atos QLM also allow advanced options like noise simulation. In this thesis we use both quantum simulators and real quantum hardware to validate our algorithms.

Operations Research and Combinatorial Optimization

Operations research (OR) is a field that provides a scientific and quantitative approach for a management to take timely and effective decisions for their problems. OR is used to define problems and to get solutions for various organizations like businesses, government organizations and non-profits. It has applications in problems like inventory control, portfolio management, production scheduling and vehicle routing, among others. All of these problems entail the maximization or minimization of a certain quantity under certain constraints. Therefore the problems in OR are intrinsically problems of optimization.

Many optimization problems are considered to be *hard*, or *NP-hard*. For these problems, we do not yet know an algorithm that solves it reasonably quickly. In mathematical terms, this means that the time taken to solve the problem increases exponentially as the size of the problem increases. We can however try to solve such NP-hard problems approximately, the idea being that we get a good enough solution in a relatively short amount of time.

There are two types of challenges while solving optimization problems: a) time - they can be hard to solve in a reasonably small amount of time, and b) size - as the size of the problems get larger, they get more and more difficult to solve. In this thesis concentrate mostly on the aspect of size. In other words, we present methods that could potentially encode very large problems on a quantum computer. Since the problems we treat are NP-Hard and are hence very difficult to solve as their size increases, the methods we propose are also approximation methods.

Shortcomings of current quantum algorithms for optimization

Despite continuous innovation in quantum hardware year upon year, current quantum computers are relatively limited in size. For example, in 2024, the largest available quantum computer on the IBM cloud is *ibm_torino*, of size 133 qubits. The largest IBM quantum computer, Condor, is of over 1000 qubits but unavailable for public use. Moreover, even on a quantum computer with 133 qubits, it is practically impossible to run an algorithm efficiently using all the qubits. This is due to several reasons, including errors like gate-error and readout-error and more importantly, the sparse connectivity among the available qubits, making it difficult to encode complex problems. The current era in quantum computing therefore is referred to as the Noisy Intermediate Scale Quantum (NISQ) era.

Within the scope of optimization applications, there has been a growing interest in quantum variational algorithms [18]–[22]. Among them, the Quantum Approximate Optimization Algorithm (QAOA) has been heavily researched [13], [14], [23]. Despite its successes, implementations of QAOA are generally limited to toy problems of very small size. This is because QAOA scales linearly with problem size. This means that an optimization problem of 200 variables requires about 200 qubits. For example, an undirected Traveling Salesman Problem (TSP) of size 20 has 190 edges and with Miller-Tucker-Zemlin (MTZ) formulation requires 191 variables. Solving this using QAOA would require 191 qubits. Addressing this scalability challenge forms the central focus of this thesis.

A qubit-efficient quantum variational algorithm for optimization

In this thesis, we first present a novel algorithm that uses logarithmically fewer qubits. Using this algorithm, which we call the LogQ encoding, an optimization problem of size 200 can be encoded using only about 8 qubits (the exact number can vary depending upon the actual problem). This approach opens an avenue for treating optimization problems of unprecedented scale on universal gate-based quantum computers.

To begin with, we demonstrate the LogQ based algorithm for the MAXIMUM CUT problem, which is a graph-based optimization problem. The problem is encoded on the quantum computer by taking inspiration from a formulation of the MAXIMUM CUT problem using the Laplacian matrix of the associated graph. The algorithm consists of several parts including the creation of the ansatz, decomposition of the Laplacian matrix into a string of Pauli terms, and finally the calculation of the expectation value. All the steps of the algorithm are explained in detail in a step-by-step calculation of a toy example.

Next, we aim to extend this algorithm to several other NP-hard problems. This is done by two approaches:

1. By converting other NP-hard problems into the MAXIMUM CUT problem using polynomial time reductions. In a 1972 paper, Richard Karp showed that we can reduce one NP-complete problem to another NP-complete problem. Recent research has brought forward a much larger set of NP-complete problems that can be connected to each other via reductions. We use this idea to convert a number other problems to the MAXIMUM CUT problems, such that finding the solution to the MAXIMUM CUT problem would be equivalent to finding the solution to the problem in question.
2. By formulating the problem as a Quadratic Unconstrained Binary Optimization (QUBO) problem. The QUBO form is a generalization of Laplacian matrix formula used in the case of the MAXIMUM CUT problem. Using this generalization, a much larger class of problems can be tackled.

The performance of the algorithm is then benchmarked for several optimization problems against classical (non-quantum) optimization methods.

A hybrid quantum-assisted algorithm for the Fleet Conversion Problem

Having described the LogQ encoding, we then concentrate on its possible industrial use-cases. The fleet conversion problem is a problem of transportation. It aims to reduce the carbon emissions and cost of operating a fleet of vehicles for a given set of tours or trips. This problem is first described as the graph coloring problem and then using the column generation technique, it is written in terms of coordinator problems and worker problems (sub-problems). We devise a hybrid method wherein the coordinator problems are solved using the commercial solver Gurobi while the worker problems are solved by the quantum solver based on the LogQ encoding. We then test our algorithm on synthetically generated instances.

Mesh segmentation using LogQ encoding

The second use-case addressed in this thesis is that of Mesh Segmentation. This project was carried out within the framework of the European Union funded project NExt ApplicationS of Quantum Computing (NEASQC) which aims to find industrial applications of near-term quantum computers. Mesh segmentation using quantum variational algorithms was one of the several use-cases identified by the NEASQC consortium.

A numerical mesh is a set of faces, edges and connecting points that can be used to describe a 2D or 3D image or model. By segmenting a mesh we attempt to dissociate the various logical parts of an image. For example, a 3D model of human can be divided into segments such as arms, head etc. The segmentation of the numerical mesh plays an important part in computer modeling, which is extensively used in the core domains of TotalEnergies' activities such as Earth imaging, physical modeling for reservoirs, and others.

In order to tackle the problem of mesh segmentation using the LogQ encoding, this problem is first converted into a graph optimization problem. The problem is equivalent to finding good clusters in a graph. A suitable objective function called modularity is defined in order to find these clusters. The maximization of this objective function is carried using LogQ. The results are then compared to KMeans, which is a classical clustering method.

Organization of the thesis

The thesis is organized as follows. Chapter [1](#) lays the foundation by introducing the basic concepts of quantum computing, optimization and contemporary quantum variational algorithms for optimization, necessary to understand all subsequent chapters. It is followed by chapter [2](#) which deals with the description and applications of the LogQ encoding and its performance benchmarks. Chapter [3](#) and chapter [4](#) deal with the two use cases: fleet conversion and mesh segmentation respectively. Finally, this thesis is concluded by summarizing our complete work and give some perspectives for further research.

This inaugural chapter is dedicated to establishing the foundational framework for the thesis. It is divided into three sections: 1) Elements of quantum computing, 2) Combinatorial optimization and 3) Variational quantum algorithms for optimization. The first section introduces the basic concepts and mathematical tools of quantum computing. This is followed by a section describing the fundamentals of combinatorial optimization and various methods to approach them. Finally, in the third section we combine the ideas of quantum computing and combinatorial optimization and talk about variation quantum algorithms for optimization, which is the main focus of this thesis.

1.1 Elements of quantum computing

In this section, we describe the basic concepts of quantum computing that are going to be used throughout this thesis. The goal of this section shall not be to present a comprehensive overview of quantum computing but rather to focus on the essential topics crucial for understanding the thesis. For a more complete understanding see [1], [2]. We begin with the qubits, the elementary building blocks of all quantum computers and then define all the elements required to carry out computation on a universal gate-based quantum computer.

1.1.1 Qubit

Much like the classical bit which forms the basis of classical computation, the quantum bit or qubit is at the heart of quantum computation. The classical bit can have 2 values, 0 and 1. The qubit on the other hand can be 0, 1 and everything in between.

More precisely, instead of having values, a qubit is said to be in a specific *state*. Two of the possible states of a qubit are $|0\rangle$ and $|1\rangle$. These are called the *basis states* or the computational basis states of a qubit. In *Dirac notation*, the symbol $|\cdot\rangle$ is called the *ket-vector* or just a ket. Dirac notation is the standard notation for quantum states and shall be used throughout this thesis. As the name suggests, all the other states of a qubit are defined by these basis states. A qubit can exist in any linear combination of the basis states. Let $|\psi\rangle$ define the state of a qubit, then it can be mathematically defined as:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \quad (1.1)$$

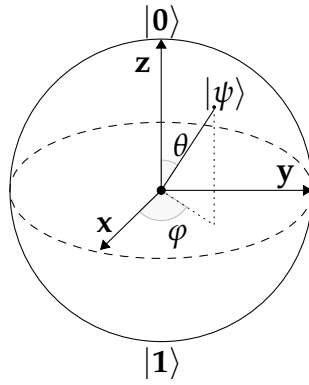


Figure 1.1: The Bloch Sphere representing a qubit in its 1-qubit Hilbert space.

where α and β are complex coefficients called the *amplitudes*. According to one of the fundamental postulates of quantum mechanics, the squared modulus of the amplitudes define the probability to measure a certain basis state. This means that $|\alpha|^2$ and $|\beta|^2$ represent the probability of a qubit being in the state $|0\rangle$ and $|1\rangle$ respectively. Since the total probability is always 1, the amplitudes are related by the equation:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (1.2)$$

This relation is called the normalization condition. The existence of qubits in all the intermediate states between 0 and 1 is called *superposition*.

The vector space created by a qubit (or a group of qubits, as we shall see in the next section) is called the *Hilbert Space*. In figure 1.1, we see a graphical representation of this space using the Bloch sphere. The qubit is in the state $|\psi\rangle = \cos\frac{\theta}{2} + e^{i\varphi}\sin\frac{\theta}{2}$. In the figure, the angles φ and θ define a point on the 3-dimensional sphere.

The state of a qubit can be represented equivalently as a column vector:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad (1.3)$$

Since the state $|0\rangle$ has $\alpha = 1$ and $\beta = 0$ and the state $|1\rangle$ has $\alpha = 0$ and $\beta = 1$, the computational basis states can be represented as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (1.4)$$

1.1.2 Multi-qubit systems

The next step is to understand how multiple qubits behave together. Much like in classical computation where we use several bits to store and process information, it is when we use several qubits together when we discover the true strength of a quantum computer.

To understand multi-qubit system it is important to define the notion of *tensor product*.

Given two column vectors $|\mathcal{V}\rangle = \begin{pmatrix} v_1 \\ \vdots \\ v_{d_v} \end{pmatrix} \in \mathbb{C}^{d_v}$ and $|\mathcal{W}\rangle = \begin{pmatrix} w_1 \\ \vdots \\ w_{d_w} \end{pmatrix} \in \mathbb{C}^{d_w}$, the tensor

product between them is defined as:

$$|\mathcal{V}\rangle \otimes |\mathcal{W}\rangle = \begin{pmatrix} v_1 \begin{pmatrix} w_1 \\ \vdots \\ w_{d_w} \end{pmatrix} \\ \vdots \\ v_{d_v} \begin{pmatrix} w_1 \\ \vdots \\ w_{d_w} \end{pmatrix} \end{pmatrix} \quad (1.5)$$

The tensor product $|\mathcal{V}\rangle \otimes |\mathcal{W}\rangle$ has a dimension of $d_v \times d_w$ and is usually denoted by the short form $|\mathcal{VW}\rangle$.

Tensor products can be carried out between matrices as well. Let there be two matrices $A \in \mathbb{C}^{p \times q}$ and $B \in \mathbb{C}^{r \times s}$. The tensor product them is defined as:

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1q}B \\ \vdots & \ddots & \vdots \\ a_{p1}B & \dots & a_{pq}B \end{pmatrix} \quad (1.6)$$

Let us start with the simplest case of multi-qubit systems. If we have 2 qubits $|\psi_1\rangle$ and $|\psi_2\rangle$ such that:

$$|\psi_1\rangle = \alpha |0\rangle + \beta |1\rangle \quad (1.7)$$

$$|\psi_2\rangle = \gamma |0\rangle + \delta |1\rangle \quad (1.8)$$

then we can define a 2 qubit system as :

$$|\psi_1\rangle \otimes |\psi_2\rangle = (\alpha |0\rangle + \beta |1\rangle) \otimes (\gamma |0\rangle + \delta |1\rangle) \quad (1.9)$$

$$= \alpha\gamma |00\rangle + \alpha\delta |01\rangle + \beta\gamma |10\rangle + \beta\delta |11\rangle \quad (1.10)$$

$$= c_0 |00\rangle + c_1 |01\rangle + c_2 |10\rangle + c_3 |11\rangle \quad (1.11)$$

where $|00\rangle = |0\rangle \otimes |0\rangle$ and so on. Generalizing from equation (1.11) we can write the N -qubit state as:

$$|\psi_N\rangle = \sum_{i=0}^{2^N-1} c_i |x_i\rangle \quad \text{where } x_i \in \{0,1\}^{\otimes N} \quad (1.12)$$

which can also be represented as:

$$|\psi_N\rangle = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2^N-1} \end{pmatrix} \quad (1.13)$$

We thus have a general N -qubit state in the computational basis. This N -qubit state resides in a $2^N \times 2^N$ dimensional Hilbert space.

A very interesting property of qubits is *entanglement*. Entanglement, in quantum mechanics, means that the state of a qubit is intrinsically linked to the state of another qubit. In other words, they cannot be separated. This can be defined mathematically as:

$$|\psi_1\psi_2\rangle \neq |\psi_1\rangle \otimes |\psi_2\rangle \quad (1.14)$$

Entanglement has a huge impact on the number of coefficients required to define a multi-qubit system. Let us take for example a 3-qubit state that is not entangled and therefore can be expressed as:

$$|\psi_3^{ne}\rangle = (c_1 |0\rangle + c_2 |1\rangle) \otimes (c_3 |0\rangle + c_4 |1\rangle) \otimes (c_5 |0\rangle + c_6 |1\rangle) \quad (1.15)$$

This state is defined by 6 coefficients. Now if we have a fully entangled 3-qubit system we will have:

$$|\psi_3^e\rangle = c_1 |000\rangle + c_2 |001\rangle + c_3 |010\rangle + c_4 |011\rangle + c_5 |100\rangle + c_6 |101\rangle + c_7 |110\rangle + c_8 |111\rangle \quad (1.16)$$

Here we require 8 coefficients to define that state. The entangled state requires more coefficients and therefore there is no way to represent $|\psi_3^e\rangle$ as a tensor product of the three qubits. In general, an N -qubit state where the qubit are completely independent require $2N$ coefficients whereas an N -qubit fully entangled state would require 2^N qubits.

The Pauli matrices are fundamental in quantum mechanics and will be used throughout this thesis. They are a set of three 2×2 matrices as follows:

Name	Symbol	Matrix
Pauli X	X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli Y	Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli Z	Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Table 1.1: List of Pauli Matrices

These three matrices are Hermitian, unitary and involutory. Together with the identity matrix (I), they form a *real vector space basis*. In other words, any 2×2 Hermitian matrix can be represented as a linear combination of the Pauli matrices and the identity matrix in such a way that all the coefficients are real.

1.1.3 Some important properties of Dirac algebra

In addition to the ket-vector, we also have the *bra-vector* which is the Hermitian conjugate of the ket. This means that as the ket is a column vector, the bra is a row vector. Its elements are the complex conjugates of the elements in the ket vector. Therefore:

$$\langle \psi_N | = |\psi_N\rangle^\dagger = (c_0^* \quad c_1^* \quad \dots \quad c_{2^N-1}^*) \quad (1.17)$$

Next, we define the concept of *inner product* where we just multiply a bra and a ket vector. If we have two states $|\psi\rangle$ and $|\phi\rangle$, the inner product is denoted as $\langle \psi | \phi \rangle$. Let us take for example the inner product of $|\psi_N\rangle$ with itself:

$$\langle \psi_N | \psi_N \rangle = (c_0^* \quad c_1^* \quad \dots \quad c_{2^N-1}^*) \begin{pmatrix} c_0 \\ c_1 \\ \cdot \\ \cdot \\ c_{2^N-1} \end{pmatrix} = c_0^* c_0 + c_1^* c_1 + \dots + c_{2^N-1}^* c_{2^N-1} \quad (1.18)$$

Using the property of complex numbers $zz^* = |z|^2$ we can thus write:

$$\langle \psi_N | \psi_N \rangle = |c_0|^2 + |c_1|^2 + \dots + |c_{2^N-1}|^2 = 1 \quad (1.19)$$

It can be easily verified that $\langle 0|0\rangle = \langle 1|1\rangle = 1$ whereas $\langle 0|1\rangle = \langle 1|0\rangle = 0$. This is the definition of *orthogonality* of a basis set.

Another important result can be obtained from the inner product $\langle 0|\psi\rangle$.

$$\langle 0|\psi\rangle = \langle 0|(\alpha|0\rangle + \beta|1\rangle) \quad (1.20)$$

$$= \alpha \langle 0|0\rangle + \beta \langle 0|1\rangle \quad (1.21)$$

$$= \alpha \quad (1.22)$$

From this we can say the probability to find the state $|\psi\rangle$ in the state $|0\rangle$ is:

$$p(0) = |\alpha|^2 = |\langle 0|\psi\rangle|^2 \quad (1.23)$$

Similarly:

$$p(1) = |\beta|^2 = |\langle 1|\psi\rangle|^2 \quad (1.24)$$

The inner products $\langle 0|\psi\rangle$ and $\langle \psi|0\rangle$ are the complex conjugates of each other. This can be shown as follows:

$$\langle \psi|0\rangle = (\alpha^* \langle 0| + \beta^* \langle 1|) |0\rangle \quad (1.25)$$

$$= \alpha^* \langle 0|0\rangle + \beta^* \langle 0|1\rangle \quad (1.26)$$

$$= \alpha^* \quad (1.27)$$

$$= \langle 0|\psi\rangle^* \quad (1.28)$$

In general:

$$\langle \psi|\phi\rangle = \langle \phi|\psi\rangle^* \quad (1.29)$$

1.1.4 Quantum Gates and Circuits

Once we have prepared our quantum state, we need to be able to perform operations on them so as to change the state. Just like we have classical gates like NOT, AND, OR, XOR etc. which alter the state of classical bit, we have quantum gates that alter the state of a qubit.

Since our N -qubit state is an $2^N \times 1$ matrix, and we want to perform an operation that changes it to another state, and hence another $2^N \times 1$ matrix, our operation is therefore a $2^N \times 2^N$ matrix. For a single qubit, we will have quantum gates as matrices of size 2×2 , for 2 qubits they will be 4×4 matrices and so on.

There are several standard gates used in gate-based quantum computing. The most commonly used ones are either 1 or 2-qubit gates. Single-qubit gates alter the state of, as their name suggests, a single qubit. For example, the Pauli-X gate can be seen as the equivalent of the classical NOT gate. In other words, it flips $|0\rangle$ to $|1\rangle$ and vice versa. Mathematically this would appear as follows:

$$X|\psi\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \alpha X|0\rangle + \beta X|1\rangle = \alpha|1\rangle + \beta|0\rangle \quad (1.30)$$

Another important single-qubit gate is the Hadamard gate. This gate creates an equal superposition of the basis states. This can be demonstrated as follows:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad (1.31)$$


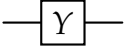

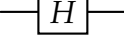

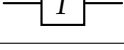
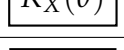
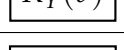
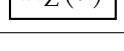
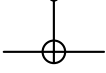
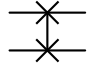
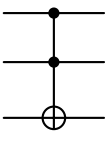
Gate	Symbol	Matrix
Pauli X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Hadamard		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Phase Gate		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
T Gate		$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
RX Gate		$\begin{pmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$
RY Gate		$\begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$
RZ Gate		$\begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
SWAP		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
Toffoli		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

Table 1.2: List of commonly used quantum gates

Both these gates, like all other single-qubit gates, create a rotation in Hilbert space. This is quite straightforward to imagine since all gates take us from one point in the Hilbert space (α, β) to another point (α^*, β^*) .

Another type of gate is the two-qubit gate. These gates are extremely important in quantum computing since these are the gates that introduce *entanglement* in the quantum state. The idea of entanglement is that the state of one qubit is going to depend on another qubit. The quintessential two-qubit gate is the controlled-NOT or the CNOT gate. In the CNOT gate there is a control qubit and a target qubit. The state of the target qubit will change *depending upon* the state of the control qubit, hence entangling them in the process.

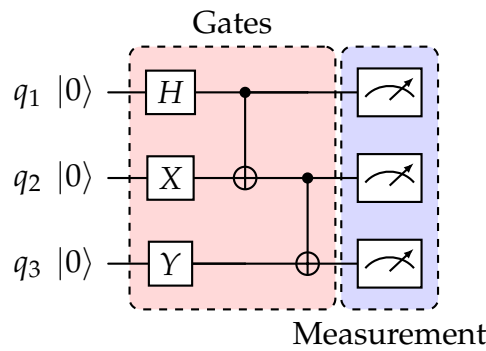
If the state of the control is $|0\rangle$ then the state of the target is left unchanged. When the state of the control is $|1\rangle$, the Pauli-X gate (i.e. the NOT gate as explained earlier) is applied

to the second qubit. Since the single-qubit gates are rotations, we can see that CNOT gate is a controlled rotation. There exists, therefore, a controlled version of every single qubit gate.

We can also have gates of over 2 qubits, but they are not as common as we usually decompose gates of more than 2 qubits into a collection of 1-qubit and 2-qubit gates.

In Table 1.2 some of the most important gates are shown.

Having both qubits and a way to alter them, we can now combine them into a quantum circuit. A quantum circuit begins with all the qubits being in the $|0\rangle$ state by convention, followed by several single-qubit and two-qubit gates. Finally all the qubits are measured to give us the result of the quantum circuit. We will speak about measurement in more detail in the following section. The diagram below demonstrates a simple quantum circuit of 3 qubits.



1.1.5 Measurement, Observables and Expectation Value

In our quantum circuit, we started with the state $|0\rangle^{\otimes N}$. After doing several operations we have reached the final state, say $|\psi\rangle$. Now we want to measure our qubits in order to know this final state.

Measurement in quantum computing is *always* with respect to a *measurement basis*. In the quantum computers, measurement is carried out in the computational basis which is also called the Z-basis. The reason why it is called the Z-basis is because the computational basis states $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ are the eigenvalues of the Pauli-Z matrix $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$.

Reading the qubits once gives a bit-string of the same length as the number of qubits. But this does not give any information regarding the state of the qubits. Rather, we need to read the qubits several times and create a probability distribution of all the possible bit-strings, as shown in figure 1.2

Since the probabilities equal the square of amplitudes, we therefore have information regarding the state of the system. This constitutes a single measurement in the Z-basis. What we can also extract from this measurement is the energy of the system.

In order to calculate this, we can represent the Z matrix as follows:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1| \quad (1.32)$$

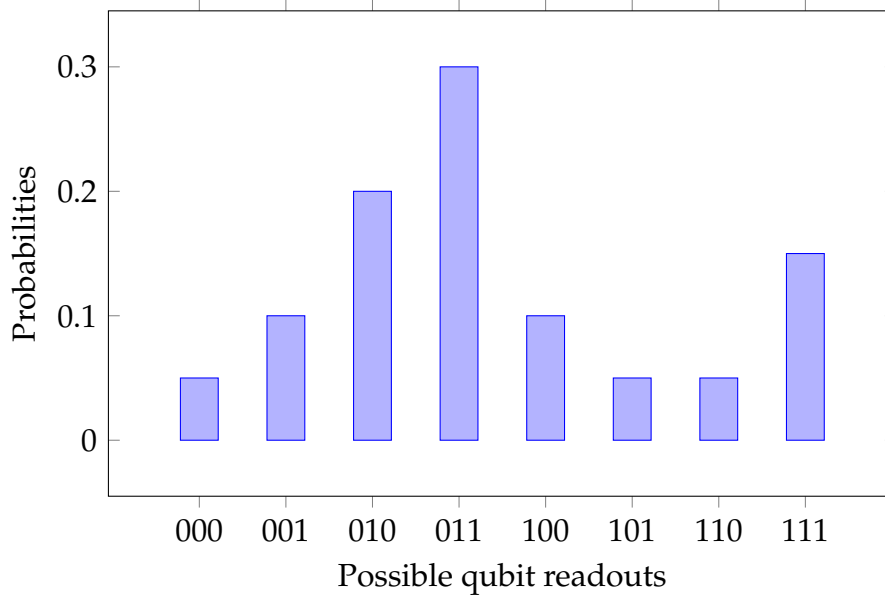


Figure 1.2: Example probability distribution for the measurement of a 3-qubit quantum circuit.

Then we can calculate $\langle \psi | Z | \psi \rangle$ as follows:

$$\langle \psi | Z | \psi \rangle = \langle \psi | (|0\rangle \langle 0| - |1\rangle \langle 1|) | \psi \rangle \quad (1.33)$$

$$= \langle \psi | 0 \rangle \langle 0 | \psi \rangle - \langle \psi | 1 \rangle \langle 1 | \psi \rangle \quad (1.34)$$

$$= \langle 0 | \psi \rangle^* \langle 0 | \psi \rangle - \langle 1 | \psi \rangle^* \langle 1 | \psi \rangle \quad (1.35)$$

$$= |\langle 0 | \psi \rangle|^2 - |\langle 1 | \psi \rangle|^2 \quad (1.36)$$

$$= p(0) - p(1) \quad (1.37)$$

Thus we get the energy of the system from the probability distribution. We can extend the above result for an N -qubit system as:

$$\langle \psi | Z^{\otimes N} | \psi \rangle = \sum_{x \in \{0,1\}^{2N}} (-1)^{h(x)} p(x) \quad (1.38)$$

where x belongs to the set of all possible bit-string combinations, $h(x)$ is the hamming weight or the number of 1s in the bit-string x and $p(x)$ is the probability of the bit-string x as found in the probability distribution.

Different measurement basis

While the default measurement basis is Z , measurement can be done in any other *complete orthogonal basis* set. A complete basis set is a set of states using which we can describe all the other states. The computational basis states, for example, form a complete basis.

An orthogonal basis set can be defined for all Hermitian matrices. Let us say we have a Hermitian matrix M . A matrix M is Hermitian when $M^\dagger = M$. A complete orthogonal basis $\{|M_1\rangle, |M_2\rangle, \dots\}$ can be defined where $|M_i\rangle$ are the *eigenvectors* of M . Hence for every eigenvector $|M_i\rangle$:

$$M |M_i\rangle = \lambda_i |M_i\rangle \quad (1.39)$$

where λ_i are the eigenvalues of M .

Any matrix M can be represented in terms of its basis as follows:

$$M = \sum_i \alpha_i |M_i\rangle \langle M_i| \quad (1.40)$$

Say we have a state $|\psi\rangle$ which is defined in Z -basis:

$$|\psi\rangle = \sum_i a_i |x_i\rangle, \quad x \in \{0, 1\}^2 \quad (1.41)$$

We can represent it in terms of M -basis as follows:

$$|\psi\rangle = \sum_i a_i |M_i\rangle \langle M_i|x_i\rangle \quad (1.42)$$

Equation (1.42) follows from the completeness theorem. The completeness theorem states that for any complete orthogonal basis set:

$$\sum_i |M_i\rangle \langle M_i| = \mathbb{I} \quad (1.43)$$

The equation (1.42) shows how we can go from one basis to another mathematically. This can be replicated on a quantum computer using a set of quantum operations or gates V .

$$V \sum_i a_i |x_i\rangle = \sum_i b_i |M_i\rangle \quad (1.44)$$

It is very important to note that changing the basis *does not* change the state at all. It is merely another way to represent the state. We can see that equation (1.41) and (1.42) represents the same state $|\psi\rangle$ using two different basis sets $|x_i\rangle$ and $|M_i\rangle$.

Let us take an example to understand this. Say we want to represent our state in the Pauli X -basis. The eigenvectors of the X matrix are $|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $|-\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

Given a state in the computational basis, $|\psi\rangle = a_1 |0\rangle + a_2 |1\rangle$, we would like to change the basis to $\{|+\rangle, |-\rangle\}$. To do this we choose the quantum operation $V = H$ where H is the Hadamard gate.

$$\begin{aligned} H|\psi\rangle &= a_1 H|0\rangle + a_2 H|1\rangle \\ &= a_1 \frac{|0\rangle + |1\rangle}{\sqrt{2}} + a_2 \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned} \quad (1.45)$$

Since $|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$, we have:

$$H|\psi\rangle = a_1 |+\rangle + a_2 |-\rangle \quad (1.46)$$

Therefore by using the gate H , we change basis from Z to X .

Calculating the expectation value of any observable

This idea of an observable can be extended to any Hermitian matrix. For a Hermitian matrix (observable) M , the expectation value can be defined as:

$$E = \frac{\langle \psi | M | \psi \rangle}{\langle \psi | \psi \rangle} \quad (1.47)$$

where $\langle \psi | \psi \rangle$ is the normalization constant.

As mentioned before, measuring the expectation value of Z means measuring in the Z -basis or computational basis. Similarly, measuring the expectation value of any observable M would mean measuring in the M basis.

Calculating the quantum operations V to convert any state from the computational basis to a general M -basis is difficult. On the other hand, converting from the computational basis to the X and Y basis are relatively straightforward. For this reason we decompose the matrix M as a sum of Pauli strings. A Pauli string is a combination of Pauli matrices for example $IIXX$, $IXXX$, $XXYY$ etc for 4 qubits. Here $IXXX = I \otimes X \otimes X \otimes X$ and so on.

Let the matrix M be of size $n \times n$. Note that n must be a power of 2. In the expression $\langle \psi | M | \psi \rangle$, when $|\psi\rangle$ is a N -qubit state, it is a 2^N element vector (equation (1.16)). Thus for mathematical consistency M is necessarily a $2^N \times 2^N$ matrix. Hence $n = 2^N$ or $N = \log_2 n$, n being the size of the matrix and N being the number of qubits.

Let $S = \{I, X, Y, Z\}^N = \{S_1, S_2, S_3, S_4\}^N$ be the set of Pauli matrices. We can consider n to be a power of 2 without any loss of generality. If the size of the observable matrix is n' which is not a power of 2, we can easily convert it to a size of $n = 2^{\lceil \log_2(n') \rceil}$, which is a power of 2. The extra space in the matrix is filled with 0's.

Consider the set $J = \{S_{i_1} \otimes S_{i_2} \dots \otimes S_{i_N} | i_1, i_2, \dots, i_N \in \{0, 1, 2, 3\}\}$ which consists of all tensor product combinations of the Pauli matrices.

Then M can be decomposed as:

$$M = \sum_{i=1}^{4^N} c_i J_i \quad (1.48)$$

The coefficients of the decomposition can be calculated using the following relation:

$$c_i = \frac{1}{n} \text{Tr}(J_i \cdot M) \quad (1.49)$$

Here the operator \cdot refers to the dot product between two matrices and $\text{Tr}()$ calculates the trace of the resulting matrix. The trace of a matrix is the sum of its diagonal elements. The observable therefore becomes:

$$M = \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot M) J_i \quad (1.50)$$

Once we have the decomposition of the observable matrix, the expectation value becomes a sum of the expectation values of all the terms.

$$\langle \psi | M | \psi \rangle = \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot M) \langle \psi | J_i | \psi \rangle \quad (1.51)$$

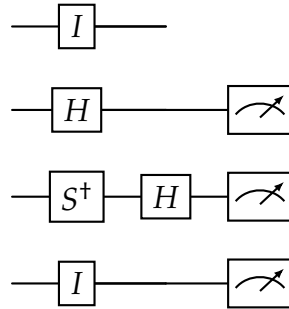
We have now successfully decomposed the expectation of M into 4^N expectation values $\langle \psi | J_i | \psi \rangle$, N being the number of qubits.

For every observable J_i , we need an operation V_i to change from the computational basis to the J_i -basis. Table 1.3 that shows the operation V required to convert from computational basis to Pauli basis:

When the measurement basis is I for a particular qubit, we do not measure that qubit. For example, if $J_i = IXYZ$, we need to add the following set of gates at the end of the circuit to convert to the required basis and calculate the expectation value:

Basis	V	Measure
X	H	Yes
Y	HS^\dagger	Yes
Z	I	Yes
I	I	No

Table 1.3: Operators to convert from computational (Z) basis to Pauli basis.



We use the operations V according to table 1.3 for the Pauli basis and apply them to the respective qubits. Once we have calculated the expectation value of every observable J_i , we can use equation (1.51) to calculate the expectation value of M . Therefore, calculating the expectation value of a matrix of size $n \times n$ requires, in the worst case, the calculation of $4^N = 4^{\log_2 n} = 2^{2 \log_2 n} = 2^{\log_2 n^2} = O(n^2)$ expectation values. For more information about the calculation of expectation values refer to [24].

1.1.6 Quantum Hardware

Until now we have discussed the theory of quantum computing. While this thesis revolves around the design of algorithms, it is also important to understand the quantum computers they are aimed at.

The algorithms in this thesis are designed for gate-based quantum computers. Gate-based quantum computers are simply quantum computers that handle the evolution of quantum states and their measurement using quantum gates. Quantum annealing, a different paradigm of quantum computing, has also been researched upon significantly but does not work on the concept of quantum gates. This paradigm of quantum computing is beyond the scope of this thesis but more details on it can be found in [25].

The aim of any quantum computer is to be able to represent qubits, or in other words, two different states of energy (or two different energy levels). In addition, it must be able to handle and control precisely the interaction between these qubits. Today (in 2024), there are several different technologies that are being used to develop quantum computers: superconducting qubits, ion-traps, neutral atoms, photonic qubits and topological qubits. Each have their own advantages and disadvantages. A detailed review of all the types of quantum hardware can be found in [1].

Superconducting qubit quantum computers are, at the moment, the most mature gate-based quantum computers. IBM has developed superconducting quantum computers which are available for use freely on their cloud. The largest size quantum computer that can be used on the cloud today is *ibm_torino* with 133 qubits. Additionally, they have announced a device with 1121 qubits, the IBM Condor, but it is not yet available for use publicly. All the experiments on quantum hardware in this thesis have been performed using IBM quantum computers.

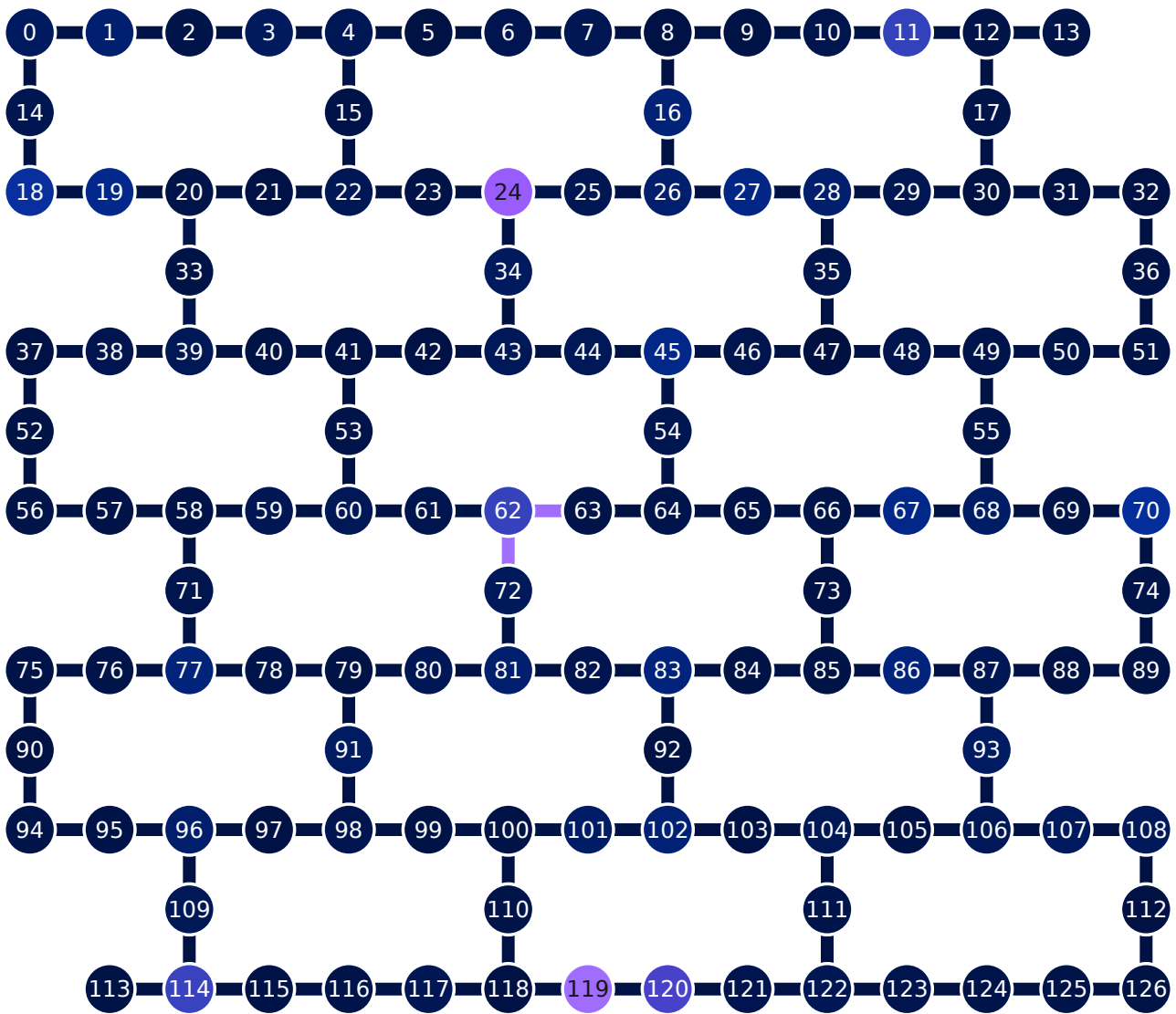


Figure 1.3: Qubit map of *ibm_brisbane* Quantum Computer

Figure 1.3 shows the qubit map of an IBM quantum computer of size 127. There are a several things to notice here. Firstly, all the qubits are not connected to each other. Qubits that are not connected to each other can still be used together using SWAP gates (see Table 1.2).

If, for example we needed to use a CNOT gate between the qubit 0 with the qubit 18, it is possible with two SWAP gates. We swap qubits 18 and 14 using the SWAP gate, meaning that qubit 14 now represents qubit 18. Now we apply the CNOT gate between the qubits 0 and 14. Finally use the SWAP gate again between qubits 14 and 18 to move them back to their original place.

If we would like to connect qubits that are far apart, like for example qubit 0 and qubit 41, it would require a lot of SWAP gates and will increase significantly the depth of the quantum circuit. Therefore, given a quantum circuit, it is important to choose the optimal qubits such that the minimum number of SWAP gates are used. It is quite clear therefore that a 127 qubit quantum circuit will not be very efficient here, given the sparse connection between them.

The colors of the qubits depict their quality. The darker the shade of the qubit, the

lower the *qubit readout error*. The readout error of `ibm_brisbane` ranges from 5×10^{-3} to 2.624×10^{-1} . A readout error of 5×10^{-3} means that out of 1000 measurements, 5 measurements are likely to be wrong. While this might seem relatively minuscule, one must take into consideration that a quantum measurement involves the measurement of a qubit several thousand times to get the probability distribution, and therefore the error becomes significant.

Similarly, for the connections in the qubit map, a darker shade of blue depicts a better quality of connection. The quality of connection is defined in terms of CNOT error. In other words, what amount of error is introduced every time a CNOT gate is applied between two qubits. The CNOT error of `ibm_brisbane` varies between 4.154×10^{-3} and 1. A CNOT error of 4.154×10^{-3} means that 0.4154% error is introduced in the circuit (and hence will reflect in the measurement) upon the application of every CNOT gate. A error of 1 (eq. between qubits 62 and 63) means that there is 100% error and hence the qubits are effectively disconnected.

CNOT errors grow in a multiplicative manner. Let's say we have 5 CNOT gates in a circuit, all with an error rate of 4.154×10^{-3} , the total error will be $1 - (1 - 4.154 \times 10^{-3})^5 = 0.02 = 2\%$. Therefore it is important of have low depth circuits with fewer CNOTs in order to get good results.

It is therefore in our interest to develop algorithms that use a low number of qubits as well as CNOTs.

1.2 Combinatorial Optimization

In the previous section we saw the elements of quantum computing. In this section we continue to build upon the components necessary to understand the thesis. Since the aim of this thesis is to tackle combinatorial optimization problems using quantum computers, we will now describe combinatorial optimization in detail.

The aim of any optimization problem is to find the minimum or the maximum value of a given multi-variable function. It is usually accompanied by either equality or inequality constraints on the variables. In combinatorial optimization, the variables in question are discrete in nature.

Let us now introduce two combinatorial optimization problems: MAXIMUM CUT and MAXIMUM INDEPENDENT SET.

MAXIMUM CUT

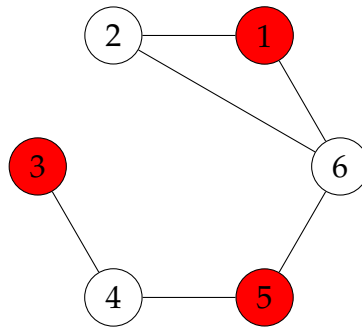
Given a weighted graph $G(V, E, w)$ where V is the set of vertices, E is the set of edges and w is the set of weights on the edges, the MAXIMUM CUT problem is defined as follows:

$$\max C(x) = \sum_{\substack{(i,j) \in E \\ i \in V, j \in V}} \frac{1}{2} w_{ij} (1 - x_i x_j), \quad x_i \in \{1, -1\} \quad (1.52)$$

The goal is to maximize the weight of the edges having vertices in different subsets. This is done by bi-partitioning the graph into two subsets such that $x_i = 1$ is one subset and $x_i = -1$ is another. The function $C(x)$ is to be maximized and is called the cost function or objective function.

Consider this graph with vertices $V \in \{1, 2, 3, 4, 5, 6\}$, edges $E \in \{(1, 2), (1, 6), (3, 4), (4, 5), (5, 6), (2, 6)\}$ and edge weights $w \in \{1, 1, 1, 1, 1, 1\}$.

It has been partitioned into two sets of nodes $\{1, 3, 5\}$ and $\{2, 4, 6\}$.



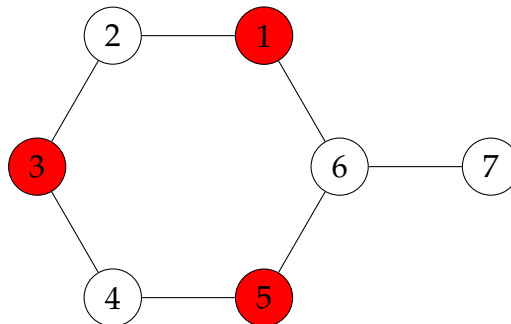
This bi-partition represents the following solution: $x = (1, -1, 1, -1, 1, -1)$. This solution means that we have $x_1 = 1$ for the first node, $x_2 = -1$ for the second node, and so on. The values of 1 and -1 define a node to be in one set or another. Putting the values of x in the equation (1.52), we get $C(x) = 5$. The aim of the optimization problem is to adjust the values x_i so as to maximize the value of $C(x)$. Note that the complementary solution $x = (-1, 1, -1, 1, -1, 1)$ gives the same result.

The total number of possible solutions for a graph with n nodes is 2^n , which is enormous. For example, for just 20 nodes, there exist over 1 million possible solutions.

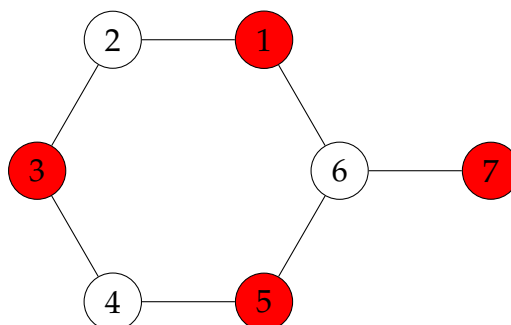
The MAXIMUM CUT problem, however does not have any constraints. This means that any solution is a *feasible* solution. Even if we choose all the nodes in one set and let the other set be a null set, it is still a solution. Most combinatorial optimization problems have constraints. A good example of a constrained problem is the MAXIMUM INDEPENDENT SET problem.

MAXIMUM INDEPENDENT SET

In a graph $G(V, E)$, an independent set is a subset of vertices $V' \subseteq V$ such that $\forall i, j \in V', (i, j) \notin E$. In other words, none of the vertices in V' have an edge between them. The figure below shows an independent set (in red).



The MAXIMUM INDEPENDENT SET problem seeks to maximize the size of the independent set. In the above graph the size of the independent set $\{1, 3, 5\}$ is 3. The maximum independent set in this case is $\{1, 3, 5, 7\}$:



A particularity of finding the maximum independent set is that the solution necessarily needs to be an independent set. While $\{1, 3, 5\}$ is not the optimal solution, it is a feasible solution. But $\{2, 3, 5\}$, for example, is not admissible as a solution as it not even an independent set, let alone the optimal one.

MAXIMUM INDEPENDENT SET can be formally defined as follows:

$$\max C(x) = \sum_{i \in V} x_i \quad (1.53)$$

$$\text{s.t. } x_i + x_j \leq 1 \quad (\forall (i, j) \in E, i \in V, j \in V) \quad (1.54)$$

$$x_i \in \{0, 1\} \quad (\forall i \in V) \quad (1.55)$$

The cost function (1.53) maximizes the size of the independent set while the constraint (1.54) makes sure that the solution is an independent set. The weighted version of this problem, MAXIMUM WEIGHTED INDEPENDENT SET, will be discussed in much more detail in the next chapters.

1.2.1 NP-Hard Problems

Computational complexity theory, a branch of computer science focusing on the resources needed to solve computational problems, introduces the concept of complexity classes such as P, NP, NP-complete, and NP-hard for *decision problems*. A decision problem is a computational problem whose answer is either *yes* or *no* (also boolean 1 and 0).

P problems are the set of problems that can be *solved* in polynomial time. This means that the time taken solve them will increase polynomially as the size of the input increases. NP problems can be *verified* in polynomial time given the *yes*-solution. It is not known whether $P = NP$ and is one of the major unsolved problems of theoretical computer science. It is widely believed that $P \neq NP$. In that case, there exist problems in NP that cannot be solved in polynomial time. NP-complete problems are problems that are at least as hard as the hardest problems in NP. In other words, they are the hardest problems that can be verified in polynomial time.

NP-hard problems are defined as problems that cannot be solved in polynomial time if $P \neq NP$. NP-Hard problems may or may not be verifiable in polynomial time. NP-complete problems are at the intersection of NP-Hard and NP. Figure 1.4 shows the Venn diagram of the complexity classes.

While, N, NP and NP-complete classes are only defined for decision problems, the definition of NP-hard extends to optimization problems as well. Since only decision problems can be NP-complete and all NP-complete problems are NP-hard as well, we can say without the loss of generality that the *optimization versions* of the decision problems are NP-hard.

In this thesis we will deal with NP-hard optimization problems. The problems of MAXIMUM CUT and MAXIMUM WEIGHTED INDEPENDENT SET, explained in the previous section, are known to be NP-hard. Their decision versions are, of course, NP-complete.

The decision version of MAXIMUM CUT is:

Input: A graph $G(V, E, w)$, V being the set of vertices, E being the edges, and $w_{ij}, (i, j) \in E$ being the weights on the edges and a positive integer k .

Decision question: Is there a set $S \subseteq V$ such that $\sum_{\substack{(i,j) \in E \\ i \in S \\ j \notin S}} w_{ij} \geq k$?

So technically we would need to solve $k + 1$ such decision problems to solve the MAXIMUM CUT optimization problem. k of these problems will have a solution of *yes* while the last problem will have a solution of *no*. The value of k for the last *yes* solution will be the solution to the optimization problem.

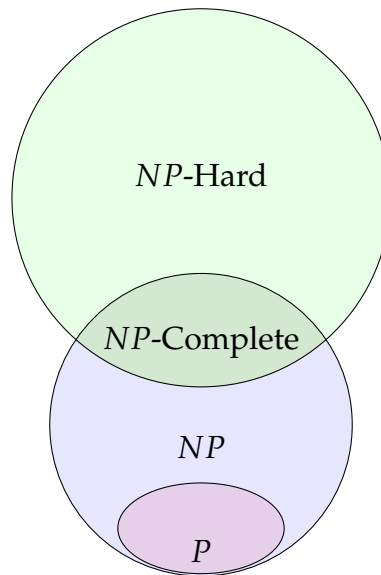


Figure 1.4: Relation between the complexity classes if $P \neq NP$

MAXIMUM CUT is one of 21 NP-complete problems shown by Karp in his paper in 1972 [26]. In this paper he showed how we can reduce one NP-complete problem to another. Reducibility of a problem \mathcal{P} to another problem \mathcal{P}' means that there exists a polynomial time algorithm to convert \mathcal{P} to \mathcal{P}' . This would imply that given that we can solve \mathcal{P}' , we can also solve \mathcal{P} by converting it to \mathcal{P}' . A more recent paper compiles a set of 296 NP-complete problems connected to each other via reductions, as shown in figure 1.5. Each node refers to one problem. For example, MAXIMUM CUT is the node 264. For more details about the problems refer to [27].

1.2.2 Integer Linear Programming

Combinatorial optimization problems can be tackled using Integer Programming. It is a description of the optimization problem where the variables are integers. In this thesis we concentrate on 0-1 Integer programming since the variables used in our problems are always binary. As can be seen in figure 1.5, 0-1 Integer Programming decision problems are known to be NP-complete, and therefore the optimization problems are NP-hard.

The simplest type of 0-1 integer programs are the Integer Linear Programs or ILPs. In ILPs the objective function as well as the constraints are linear with respect to the binary variable. An ILP with n variables and m constraints can be written in the following general form:

Objective: $\max \sum_{i=1}^n c_i x_i$

Constraints:

1. $\sum_{ij} a_{ij} x_i \leq b_j, i \in \{1, 2, 3 \dots n\}, j \in \{1, 2, 3 \dots m\}$
2. $x_i \in \{0, 1\}$

where the constants $a, b, c \in \mathbb{R}$.

A common strategy to tackle ILP problems is to relax the binary variables $x_i \in \{0, 1\}$ such that they become continuous variables between 0 and 1: $x_i \in [0, 1]$. This is called a

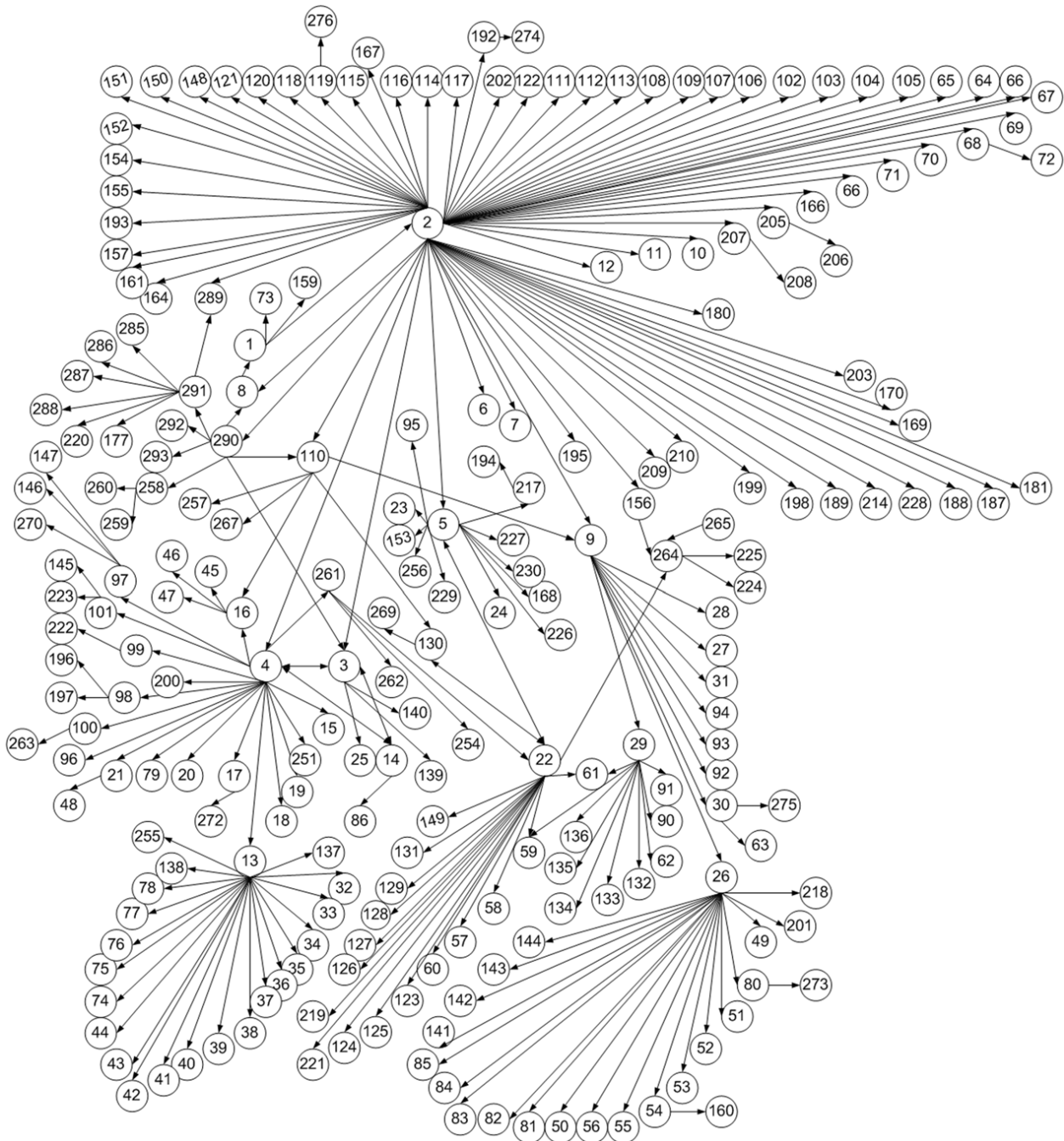


Figure 1.5: Digraph of transformations between NP-Hard problems (taken from [27]).

linear programming integrality relaxation and the resulting problem is a linear program or LP.

Linear programming is an optimization problem involving continuous variables, where both the objective function and the constraints are linear in terms of the input variables. Although LP does not strictly fall under combinatorial optimization, it is widely used as a subroutine in various exact and approximate algorithms.

Various methods exist to solve the LPs. The *simplex algorithm*, proposed by George Dantzig in 1947, is one of the most popular approaches to LP problems. A detailed description of this algorithm can be found in [28]. It is known that LP is in P (polynomially

solvable).

Since the LP relaxes the variables to continuous variables, the LP solution consists of fractional values. For a maximization problem, the optimal LP solution is the upper bound to the original ILP. The percentage gap between the optimal LP solution and the optimal ILP solution is called the *integrality gap*.

In order to generate an integer solution from the LP solution, the *branch-and-bound* method can be used. A detailed description of branch-and-bound can be found in [29].

ILPs can be solved using commercial solvers like *Gurobi* or *cplex*. These solvers use the simplex method, branch-and-bound and a variety of other methods to solve ILP problems. Note that these problems are NP-hard and therefore we do not necessarily have an optimal solution.

1.2.3 Approximation Algorithms

A problem being NP-Hard usually means that it is not possible to find the optimal solution of the problem in a reasonable amount of time. These problems, therefore, need to be approached in another way. Instead of trying to find the optimal solution, we can try to find a near-optimal solution relatively quickly. The solution hence found is thus *approximate*. An algorithm that generates, in polynomial time, an approximate solution to an optimization problem is called an *approximation algorithm*. Approximation algorithms may or may not have a theoretical bound on their performance.

The ratio between the approximate solution and the optimal solution is known as approximation ratio α .

$$\alpha = \frac{\text{Approximate Solution}}{\text{Optimal Solution}} \quad (1.56)$$

For a maximization problem, α varies from 0 to 1, with a $\alpha = 1$ denoting that the algorithm found the exact solution. For a minimization problem, $\alpha \geq 1$. It is of course not possible to find α unless we know the optimal solution of the optimization problem. This quantity, however, is important in the formal analysis of an algorithm and can be used to derive a performance guarantee.

A good example of an approximation algorithm with a performance guarantee is the Christofides algorithm [30] for the traveling salesman problem, which is a minimization problem. The algorithm has a bound of $\alpha = 1.5$.

1.2.4 Quadratic Unconstrained Binary Optimization (QUBO)

QUBO is a way to model an optimization problem in such a way that the objective function is quadratic and is free of any constraints. Note that if and only if the variables are binary, QUBO also allows for linear terms. This is because if $x_i \in \{0, 1\}$, $x_i = x_i^2$, therefore all the linear terms can be converted into quadratic terms without the loss of generality.

A general QUBO optimization model has the following form:

$$\max \sum_{ij} c_{ij} x_i x_j \quad (1.57)$$

The above equation can also be written in matrix form as follows:

$$\max (x_1 \ x_2 \ \dots \ x_n) \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \quad (1.58)$$

and hence eventually

$$\max x^T Q x \quad (1.59)$$

$$Q = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix} \quad (1.60)$$

The matrix Q is called the QUBO matrix.

While the QUBO formulation of a problem does not contain any constraints, it is nevertheless possible to encode constraints in the objective function itself. Equality constraints are modeled as a penalty terms. Let there be a maximization problem with the objective function f where the constraints are given as follows:

$$\sum_{ij} a_{ij} x_i = b_j \quad (1.61)$$

Then, in order to integrate the constraints into the objective function f , we write it as:

$$\max [f - p(\sum_{ij} a_{ij} x_i - b_j)^2] \quad (1.62)$$

Here $p \in \mathbb{R}^+$ is the strength of the penalty.

Note that if this a minimization problem with the same constraint, we need to add the penalty instead of subtracting it.

$$\min [f + p(\sum_{ij} a_{ij} x_i - b_j)^2] \quad (1.63)$$

Inequality constraints can also be handled by various methods. For more information refer to [31].

1.2.5 Goemans Williamson Algorithm

The Goemans Williamson (GW) algorithm [32] is an important approximation algorithm for the MAXIMUM CUT problem. Since it is an approximation algorithm, it does not find an exact solution but it is very efficient and has a theoretical lower bound on the solution. GW is a semidefinite programming (SDP) algorithm. In SDP, the problem is stated in terms of positive semidefinite matrices. It is a type of continuous optimization.

A matrix A is defined to be positive semidefinite ($A \succeq 0$) if and only if for a nonzero vector x :

$$x^T A x \geq 0 \quad (1.64)$$

Positive semidefinite matrices have non-negative eigenvalues. Just like non negative numbers, positive semidefinite numbers can have a square root. Therefore, a positive semidefinite matrix A can have a square root S such that:

$$S^T S = A \quad (1.65)$$

This property is useful while defining the MAXIMUM CUT problem using positive semidefinite matrices. To begin, let us recall the objective function of MAXIMUM CUT from equation (1.52).

$$C(x) = \sum_{\substack{(i,j) \in E \\ i \in V, j \in V}} w_{ij} \frac{1 - x_i x_j}{2}, \quad x_i \in \{-1, 1\} \quad (1.66)$$

The first step of the GW is to do an SDP *relaxation* of the variables. The variables x_i can be seen as being 1-dimensional vectors of unit-norm. In the SDP relaxation we allow x_i to be multidimensional vectors z_i of unit-norm. More precisely, we assume that the vectors z_i belong to the $|V|$ -dimensional unit-sphere $\mathbb{S}^{|V|}$, where $\mathbb{S}^{|V|} = \{y \in \mathbb{R}^{|V|} \mid \|y\| = 1\}$. This mapping is $z_i \in \mathbb{S}^{|V|}$ is bi-directional. Thus, given a matrix Z , we can recover the vectors z_i . This is explained in detail in the Chapter 1 of [33].

The matrix $z z^T$ represents a positive semidefinite matrix $Z \succcurlyeq 0$ where z is the square root of Z . The elements of the Z are $Z_{i,j} = z_i^T z_j$.

Replacing $x_i x_j$ by $z_i^T z_j$ in equation (1.66), we have:

$$C(z) = \sum_{\substack{(i,j) \in E \\ i \in V, j \in V}} w_{ij} \frac{1 - z_i^T z_j}{2}, \quad z_i \in \mathbb{S}^{|V|} \quad (1.67)$$

The constraint $z_i \in \mathbb{S}^{|V|}$ can be represented as $z_i^T z_i = Z_{i,i} = 1$ (see Chapter 1 in [33]). Rewriting the problem in terms of the matrix Z , we have:

$$\begin{aligned} \max C(Z) &= \sum_{\substack{(i,j) \in E \\ i \in V, j \in V}} w_{ij} \frac{1 - Z_{i,j}}{2} && \text{(GW)} \\ \text{s.t. } Z_{i,j} &\succcurlyeq 0 && (\forall i \in V, j \in V) \quad \text{(GW-1)} \\ Z_{i,i} &= 1 && (\forall i \in V) \quad \text{(GW-2)} \end{aligned}$$

From the optimal solution of this problem Z^* we can recover the optimal vectors z_i^* and hence extract a solution $x^* \in \{-1, 1\}^{|V|}$ to our problem. This is done using a randomly generated hyperplane that cuts the sphere $\mathbb{S}^{|V|}$ in two. More precisely, this hyperplane is the vector $h \in \mathbb{R}^{|V|}$ and we can recover the two partitions as:

$$x^* = \begin{cases} 1 & \text{if } h^T z_i^* \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (1.68)$$

The GW algorithm for the MAXIMUM CUT problem provides solutions that are always at least $\alpha = \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos \theta} = 0.87856$ times the optimal solution. This is proven in [32].

1.3 Variational Quantum Algorithms for Optimization

While quantum computers have shown significant promise for the future, a true quantum speedup for practical applications is yet to be realized. We are currently in the Noisy Intermediate-Scale Quantum (NISQ) era where the quantum computers are limited in terms of number of qubits, connectivity and errors that limit quantum circuit depth. Since fault tolerant quantum computers (FTQCs) could be years or even decades away, it is imperative

that we research methodologies that could help us use the NISQ computers of today. Variational quantum algorithms (VQAs) have emerged as one of the foremost strategies to take advantage of NISQ devices.

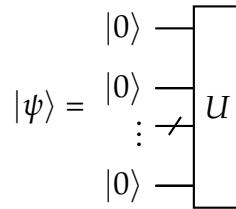
VQAs provide a general structure to solve a variety of problems, including but not limited to optimization problems. VQAs are based on the well-known variational method in quantum mechanics. In quantum physics, we are often interested in finding the ground state (minimum energy) of a quantum system. The variational method is a method to approximately find the ground state energy of a quantum state. It consists of taking a initial quantum state (usually called the trial state), which depends on one or more parameters. Then, by optimizing these parameters, the approximate ground state of the system is found. Since this is a problem of minimization, any solution found shall be an upper bound of the ground state energy.

Similarly, in a VQA we have a set of quantum operations (including the preparation of a quantum state followed by the expectation value measurement of a certain observable) that can be controlled using a set of parameters. Then, by optimizing these parameters, we can minimize the expectation value. In the context of an optimization problem having a certain cost function, the quantum operations need to be such that minimizing the expectation value is equivalent to minimizing the cost function. While we are only talking about minimization here to be coherent with the concept of finding the ground state, this does not lose generality as a maximization problem can be easily posed as a minimization problem using a ‘-’ sign.

In order to optimize the parameters, we use classical black-box optimizers. In every iteration we alternate between a classical optimizer and the quantum computer. VQAs are therefore hybrid quantum-classical algorithms.

The general structure of a VQA is as follows:

1. An initial quantum state $|\psi\rangle$, also known as an ansatz, is prepared on the quantum computer using a quantum circuit. To create the ansatz we start with the the N -qubit state $|0\rangle^{\otimes N}$ and then apply a set of quantum gates (say U).



2. The expectation value of an observable H , $\langle\psi|H|\psi\rangle$ is then calculated on the quantum computer. This can also be referred to as performing a measurement in the H basis as described in section [1.1.5](#).
3. The expectation values hence calculated must depend on certain parameters. To that end, the ansatz or the observable or both can be parameterized. Hence the expectation value can be written as:

$$E(\boldsymbol{\theta}, \boldsymbol{\phi}) = \langle\psi(\boldsymbol{\theta})|H(\boldsymbol{\phi})|\psi(\boldsymbol{\theta})\rangle \quad (1.69)$$

where $\boldsymbol{\theta} \in \mathbb{R}^p$, $\boldsymbol{\phi} \in \mathbb{R}^q$ are vectors of size p and q .

4. In the classical black-box optimizer input the initial values of the parameters $(\boldsymbol{\theta}_0, \boldsymbol{\phi}_0)$ as well as the cost function (a function that calculates the expectation value $E(\boldsymbol{\theta}, \boldsymbol{\phi})$). The parameters of the expectation value are then adjusted in an iterative manner until the expectation value is minimized.

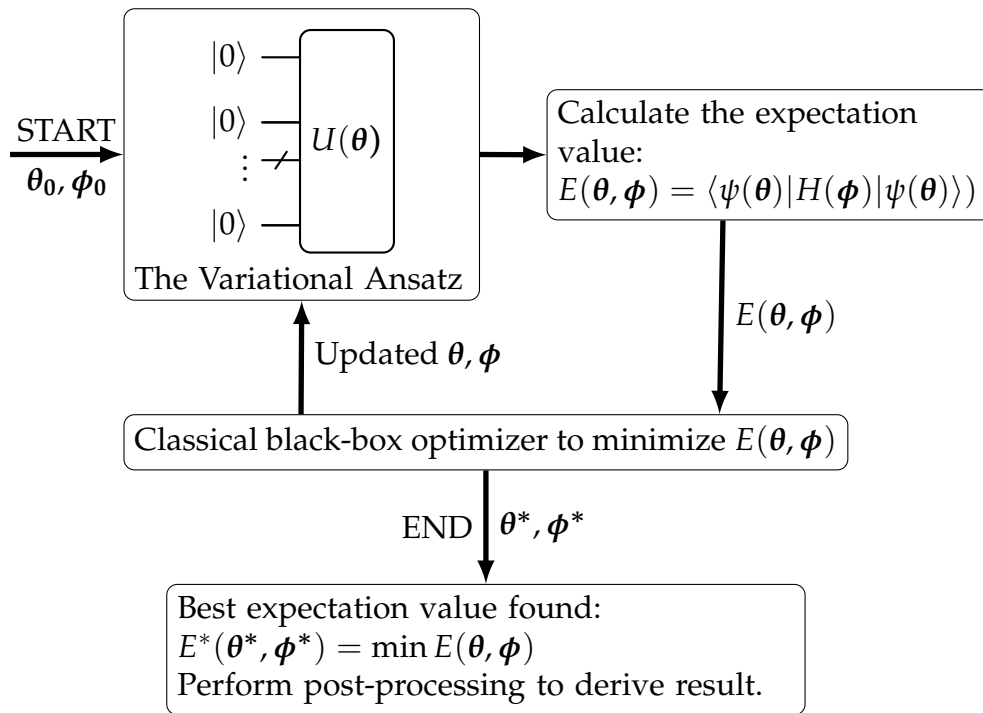


Figure 1.6: Diagrammatic representation of a VQA.

Figure 1.6 shows the general VQA structure diagrammatically.

1.3.1 Black-box optimizers

A important part of the VQAs are the classical black-box optimizers that are used to optimize the parameters. The efficiency of the black-box optimizers play a significant role in the performance of the VQA. Here, we will speak briefly about two different black-box optimizers that are used in this thesis: COBYLA and Genetic Algorithm. They are just two examples of black-box optimizers used in VQAs. For a more complete outlook on these classical optimizers refer to [18], [34].

COBYLA

Constrained Optimization BY Linear Approximations or COBYLA is a derivative-free optimization algorithm that operates without gradient information. It was proposed by M. J. D. Powell in 1994 [35]. It's particularly useful when gradients are unavailable or expensive to compute. This is helpful in the case of VQAs, where the function to optimize consists (at least in part) of quantum circuits and therefore not a mathematical function whose derivative can be easily calculated.

COBYLA uses linear approximations to represent the cost function and the constraints. These approximations are iteratively refined to converge to an optimal solution. Additionally, COBYLA uses *trust regions* to make sure that the linear approximations remain valid. The trust region defines a neighborhood around the current iterate (current point in the solution space where the algorithm is located) within which the linear model is trusted to be a reasonable approximation of the actual functions.

COBYLA is deterministic, meaning that given a specific input (parameters and objective function), it always generates the same result. It accepts real numbers as parameters and parameter bounds need to be defined as constraints. It is known to handle both equality and inequality constraints efficiently.

Genetic Algorithm

The Genetic Algorithm or GA is an evolutionary algorithm that takes inspiration from natural selection and genetics. It is very useful for complex optimization problems where the search space is large. Unlike COBYLA, GA is non-deterministic and incorporates randomness during the algorithm run, leading to different outcomes across different runs, even with the same initial conditions.

The genetic algorithm starts with a *population* of random candidate solutions to the optimization problem. The size of the population considered is defined as an input to the algorithm. It proceeds to *evolve* the population over several iterations or rather *generations*. The process of evolution of the candidate solutions is based on specific selection processes based on several algorithm parameters. This selection process determines the following generations of candidate solutions.

After several generations, the best solution is taken from all the candidates throughout the generations. GAs are heuristics and therefore do not guarantee that the algorithm will generate an optimal solution. The solution achieved therefore depends on the maximum number of allowed generations, which is an input of the GA algorithm. For a more detailed description of the GAs, refer to [36].

1.3.2 Quantum Approximate Optimization Algorithm

In this section we will talk about the Quantum Approximate Optimization Algorithm (QAOA), one of the most studied VQAs for combinatorial optimization problems. The QAOA is a hybrid quantum-classical optimization algorithm introduced by Edward Farhi and Jeffrey Goldstone in 2014 [13]. QAOA uses a discretized simulation of *Adiabatic Quantum Computation* [37].

A general combinatorial optimization problem can be defined using n bits and m binary clauses. The clauses are either *True* or *False* (1 or 0) and the objective is to find the optimal binary vector of size n to maximize the number of *True* clauses. The optimization problem is therefore:

$$\max C(\mathbf{x}) = \sum_{i=1}^m C_i(\mathbf{x}) \quad (1.70)$$

where $\mathbf{x} = \{x_1, x_2 \dots x_n\}$. This problem is generally referred to as the MAXIMUM SATISFIABILITY problem. Following are the components of the QAOA algorithm for a general combinatorial optimization problem.

The Variational Ansatz

The first step of a VQA is to create an ansatz. In the QAOA ansatz, we have three parts: the initial state, the cost operator and the mixer operator. The number of qubits required depends on the size of the problem. It is usually a linear function of the problem size. Here we will assume that we require N -qubits to carry out the QAOA algorithm.

Initial State: The first component of the variational ansatz is an initial state. The initial state should be trivial to prepare. For QAOA this is a uniform superposition of all the computational basis states. This is done by applying Hadamard gates on all the qubits.

$$|\psi_0\rangle = \begin{array}{c} |0\rangle \text{---} \boxed{H} \text{---} \\ |0\rangle \text{---} \boxed{H} \text{---} \\ \vdots \text{---} \text{---} \\ |0\rangle \text{---} \boxed{H} \text{---} \end{array}$$

$$|\psi_0\rangle = H^{\otimes N} |0\rangle^{\otimes N} = \frac{1}{\sqrt{2^N}} \sum_x |x\rangle, \quad \forall x \in \{0,1\}^N \quad (1.71)$$

Cost Operator: The next part is the unitary cost operator U_C which encodes information regarding the objective function. This is created using a cost Hamiltonian which encodes $C(\mathbf{x})$ as an operator H_C that is diagonal in computational basis, such that for an N -qubit state $|\psi\rangle$:

$$H_C |\psi\rangle = C(\mathbf{x}) |\psi\rangle \quad (1.72)$$

Using the cost Hamiltonian, we create the cost operator as follows.

$$U_C(\gamma) = e^{-i\gamma H_C} \quad (1.73)$$

where $\gamma \in [0, 2\pi]$ is a parameter or angle.

Mixer Operator: Finally we have the mixer operator U_B . Similar to the cost operator, the mixer operator contains the mixer Hamiltonian H_B . The mixer operator is required to explore the entire solution space. While the cost operator helps in calculating the cost function, the mixer operator helps the algorithm to move between the possible solutions. In QAOA the mixer Hamiltonian is mathematically defined as:

$$H_B = \sum_{j=1}^n X_j \quad (1.74)$$

where X_j is the Pauli-X operator acting on the j^{th} qubit. The mixer operator is therefore:

$$U_B(\beta) = e^{-i\beta H_B} = \prod_{j=1}^n e^{-i\beta X_j} \quad (1.75)$$

where $\beta \in [0, \pi]$ is a parameter.

Ansatz: Using the initial state, the cost and the mixer operator (using equations (1.71), (1.73) and (1.75)), we have the following ansatz:

$$|\gamma, \beta\rangle = \begin{array}{c} |0\rangle \text{---} \boxed{H} \text{---} \\ |0\rangle \text{---} \boxed{H} \text{---} \\ \vdots \text{---} \text{---} \\ |0\rangle \text{---} \boxed{H} \text{---} \end{array} \begin{array}{c} \boxed{U_C(\gamma)} \\ \boxed{U_B(\beta)} \end{array}$$

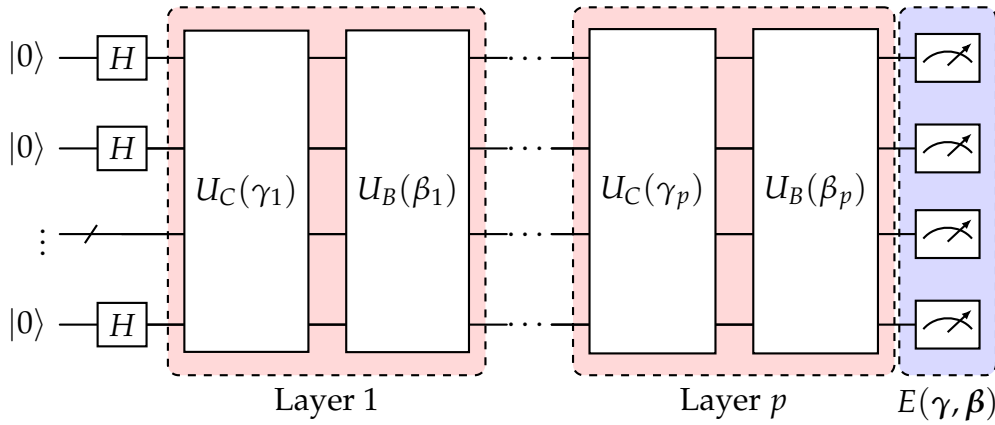


Figure 1.7: Circuit to build the QAOA ansatz $|\gamma, \beta\rangle$ and to calculate the expectation value $E(\gamma, \beta)$.

$$|\gamma, \beta\rangle = e^{-i\beta H_B} e^{-i\gamma H_C} |\psi_0\rangle \quad (1.76)$$

Multi-layer Ansatz: The above ansatz is the *single-layer* QAOA ansatz. In fact, QAOA can have p -layers. In every layer the cost and mixer Hamiltonians are applied repeatedly and with different parameters. The p -layer QAOA ansatz is therefore:

$$|\gamma, \beta\rangle = |\gamma_1 \dots \gamma_p, \beta_1 \dots \beta_p\rangle = e^{-i\beta_p H_B} e^{-i\gamma_p H_C} \dots e^{-i\beta_2 H_B} e^{-i\gamma_2 H_C} e^{-i\beta_1 H_B} e^{-i\gamma_1 H_C} |\psi_0\rangle \quad (1.77)$$

Calculating the expectation value

Once the ansatz is prepared, the expectation value of the cost Hamiltonian H_C is to be measured, giving us the value of the cost function:

$$E(\gamma, \beta) = \frac{\langle \gamma, \beta | H_C | \gamma, \beta \rangle}{|\langle \gamma, \beta | \gamma, \beta \rangle|^2} \quad (1.78)$$

Note that H_C is diagonal so it can be described in the computational basis and hence it suffices to simply measure the circuit in Z or computational basis. For a N -qubit system, the above equation therefore becomes:

$$E(\gamma, \beta) = \frac{\langle \gamma, \beta | Z^{\otimes N} | \gamma, \beta \rangle}{|\langle \gamma, \beta | \gamma, \beta \rangle|^2} \quad (1.79)$$

The quantum circuit to calculate $E(\gamma, \beta)$ is shown in figure [1.7](#).

Optimizing the expectation value

Finally, we maximize the cost function using a classical black-box optimizer such as a genetic optimizer or COBYLA as explained previously. Each loop of the classical optimizer consists of calculation an expectation value. QAOA is a heuristic algorithm and therefore attempts to find an approximate solution to the optimization problem. There is, therefore no proof of optimality and we take the best solution found.

$$E^* = \max E(\gamma, \beta) \quad (1.80)$$

The solution to the problem is then extracted from the probability distribution of the quantum state at expectation value E^* .

QAOA for the MAXIMUM CUT problem

For a given problem, we only need to define the cost Hamiltonian and then the other steps of the algorithm are fixed. For the MAXIMUM CUT problem on a graph $G(V, E, w)$ as defined in (1.52) the cost Hamiltonian is:

$$H_C = \sum_{(i,j) \in E} \frac{w_{ij}}{2} (1 - Z_i Z_j) \quad (1.81)$$

Since the QAOA algorithm was proposed in 2014, there has been a significant amount of research carried out on the topic as there have been several extensions to QAOA. A detailed review of several such extensions can be found in [38]. A detailed mathematical description and analysis of QAOA can be found in [39].

The LogQ Encoding: Solving NP-Hard problems using exponentially fewer qubits

2.1 Introduction

In chapter 1 we have seen the fundamental concepts of quantum computing, combinatorial optimization and variational quantum algorithms. In this chapter, we combine these ideas to present a novel quantum variational algorithm for a plethora of NP-hard combinatorial optimization problems.

As explained in chapter 1, NP-hard optimization problems are problems that do not have algorithms that can give an exact solution in polynomial time, whereas it is 'easy' to verify the solution if it is known [40]–[42]. While finding exact solutions to large problems is difficult, there exist many algorithms that find approximate solutions to these problems [43]–[46]. In the scope of quantum computing, a huge amount of research has been carried out on hybrid quantum-classical algorithms [13], [14], [18]–[22], [47]–[52]. In such algorithms, quantum circuit measurements are used in tandem with a classical optimization loop to obtain an approximate solution.

The Quantum Approximate Optimization Algorithm (QAOA) [13], [23], [53]–[55], explained in detail in the previous chapter, is one of the most studied hybrid quantum-classical algorithm. The QAOA, however, has a significant drawback that limits its scope of application severely. As we increase the size of the optimization problem, the number of qubits required to solve the problem using QAOA increases linearly [56]. This is a problem for several reasons. Firstly, at the moment (mid-2024), the largest announced universal gate-based quantum computer is IBM's Condor device, containing 1121 qubits. The largest device available on their cloud-based platform is the IBM Torino having 133 qubits. Secondly, all the qubits are not of the same quality and the larger the problem, the more likely it is to obtain noisier results due to the presence of qubits with higher error rates. Thirdly, these qubits are not all-in-all connected, meaning that in case of large sized problem, numerous SWAP gates would have to be used in order to run the circuit, leading to more noise.

It is in this context that we introduce a new way to encode combinatorial optimization problems on quantum computers [57]. This encoding, which we call the *LogQ encoding*, allows us to represent much larger problems using a fairly small number of qubits. Therefore a MAXIMUM CUT problem with a graph of n nodes can be represented using only $\lceil \log_2 n \rceil$

qubits.

While the LogQ encoding, like the QAOA, is a variational quantum algorithm, it is important to note that they are fundamentally different. While QAOA uses the Hilbert space to expand in an exponential manner all of the 2^n solution space at once, LogQ exploits the *compression* power of the Hilbert space in a logarithmic way.

We start with describing the method introduced in [57] which is the LogQ encoding for the MAXIMUM CUT problem. We then extend this algorithm to several other NP-hard problems. This is approached in two different ways, as demonstrated in the following sections.

The chapter is structured as follows. In section 2.2.1, we describe in detail the LogQ encoding of the MAXIMUM CUT problem on a quantum computer. In section 2.2.2, we show how this algorithm can be applied on a variety of NP-hard problems by converting them, directly or indirectly, to the MAXIMUM CUT problem. In section 2.2.3, we show how any Quadratic Unconstrained Binary Optimization Problem (QUBO) problem can be treated using the LogQ encoding. In section 2.3, experimental results of all the methods described in the previous sections are shown. Notably, we show quantum simulator results with instances of sizes of over a hundred nodes/objects, as well as quantum hardware (QPU) results for problem sizes up to 256.

This chapter is partly adapted from our work published in Physical Review A [58]. This work was also presented in conferences ROADEF 2023 [59] and EU/ME conference 2023 [60].

2.2 Methods

2.2.1 A qubit-efficient MAXIMUM CUT Algorithm

Contemporary quantum optimization algorithms in general scale linearly with problem size. This means that if the problem consists of an n node graph, the algorithm will require n qubits to solve the problem. Note that to solve a problem here implies to obtain an approximate solution. Following Ref. [57], we present an algorithm that scales logarithmically with the problem size. For a problem of size n , the number of qubits required is $\lceil \log_2 n \rceil$.

2.2.1.1 Description of the algorithm

Recall first the definition of MAXIMUM CUT:

MAXIMUM CUT

Input A weighted graph $G(V, E, w)$.

Task Find $x \in \{1, -1\}^{|V|}$ that maximizes $\sum_{ij} w_{ij} \frac{1 - x_i x_j}{2} \forall \{(i, j) \in E\}$, where w_{ij} are the weights on the edges.

Given a graph $G(V, E)$, the MAXIMUM CUT can be represented using the Laplacian matrix. The Laplacian matrix is defined as follows:

$$L_{ij} = \begin{cases} \text{degree}(i) & \text{if } i = j \\ -w_{ij} & \text{if } i \neq j \text{ and } (i, j) \in E \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

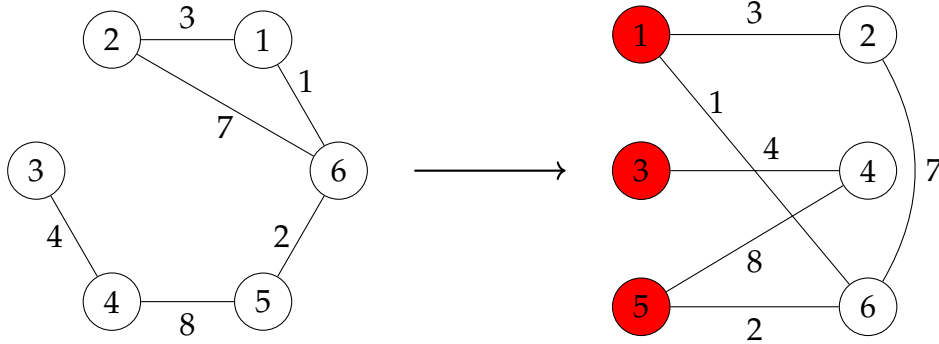
Note that the degree here refers to the weighted degree. The weighted degree of a vertex is the sum of weights of all edges containing the vertex.

The MAXIMUM CUT value is given by the following equation [61]:

$$C(x) = \frac{1}{4}x^T Lx \quad (2.2)$$

where L is the Laplacian matrix and $x \in \{1, -1\}^{|V|}$ is the bi-partition vector.

To see this let us take an example. Consider a graph $G(V, E, w)$ such that $V \in \{1, 2, 3, 4, 5, 6\}$, $E \in \{(1, 2), (2, 6), (3, 4), (4, 5), (5, 6), (6, 1)\}$ and $w \in \{3, 7, 4, 8, 2, 1\}$ such that the weight of the edge $(1, 2)$ is 3 as so on.



The Laplacian matrix of this graph is:

$$L = \begin{bmatrix} 4 & -3 & 0 & 0 & 0 & -1 \\ -3 & 10 & 0 & 0 & 0 & -7 \\ 0 & 0 & 4 & -4 & 0 & 0 \\ 0 & 0 & -4 & 12 & -8 & 0 \\ 0 & 0 & 0 & -8 & 10 & -2 \\ -1 & -7 & 0 & 0 & -2 & 10 \end{bmatrix} \quad (2.3)$$

The graph is partitioned into 2 sets, the dark and the light colored vertices. The vector x for this bi-partition can be written as 101010, putting the dark vertices as 1 and the light vertices as 0. Note that we can also take the complimentary vector 010101 as it describes the same bi-partition. Putting the values of x and L in equation (2.2), we have:

$$C = [1 \ 0 \ 1 \ 0 \ 1 \ 0] \begin{bmatrix} 4 & -3 & 0 & 0 & 0 & -1 \\ -3 & 10 & 0 & 0 & 0 & -7 \\ 0 & 0 & 4 & -4 & 0 & 0 \\ 0 & 0 & -4 & 12 & -8 & 0 \\ 0 & 0 & 0 & -8 & 10 & -2 \\ -1 & -7 & 0 & 0 & -2 & 10 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad (2.4)$$

$$= [1 \ 0 \ 1 \ 0 \ 1 \ 0] \begin{bmatrix} 4 \\ -3 \\ 4 \\ -12 \\ 10 \\ -3 \end{bmatrix} \quad (2.5)$$

$$= 18 \quad (2.6)$$

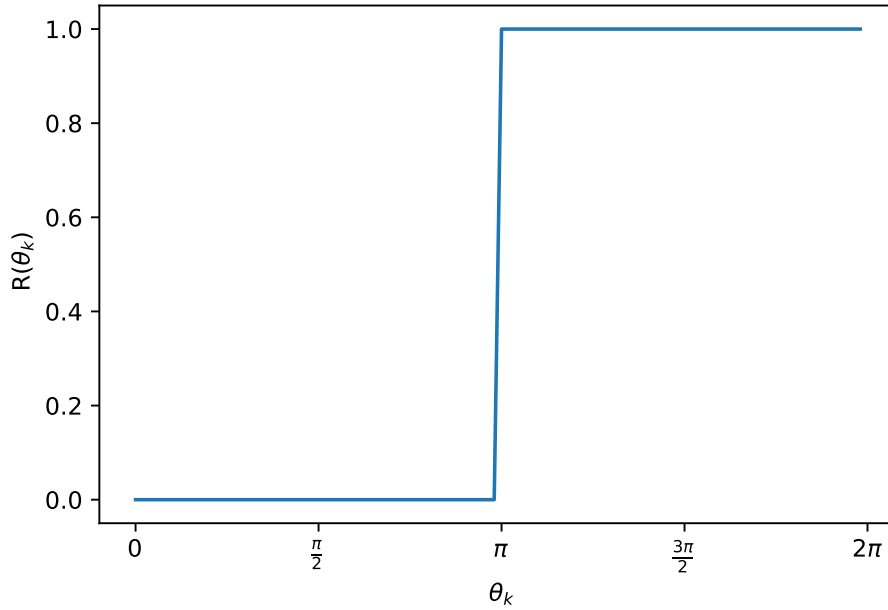


Figure 2.1: The function R

It is quite straightforward to verify that the cut in the given bi-partition is indeed 18. The edges with different vertex colors are $(1, 2), (1, 6), (3, 4), (4, 5), (5, 6)$ whose weights add up to $3 + 1 + 4 + 8 + 2 = 18$.

Due to fact that the Laplacian is a Hermitian matrix, it resembles a Hamiltonian of an actual physical system. The quantum analog of equation (2.2) is

$$C(\theta_1 \dots \theta_n) = 2^{N-2} \langle \Psi(\theta_1 \dots \theta_n) | L | \Psi(\theta_1 \dots \theta_n) \rangle \quad (2.7)$$

where L is the Laplacian matrix of the graph, $|\Psi\rangle$ is the parameterized ansatz and $\theta = \{\theta_1 \dots \theta_n\}$ are the variables to be optimized. If the size of the graph is n , then L is of size $n \times n$ and $|\Psi\rangle$ is a state of $N = \lceil \log_2 n \rceil$ qubits. 2^{N-2} is the normalization constant.

We have designed a variational algorithm that finds a good approximation to the best MAXIMUM CUT. Starting from the initial values of θ parameters, we call a quantum circuit to evaluate the objective function (Algorithm 1) and run a classical black box optimization loop over the θ parameters (Algorithm 2). As a result, we obtain θ^* to evaluate the best solution.

To evaluate the expectation value C on a quantum computer, first we need to create the ansatz $|\Psi(\theta_1 \dots \theta_n)\rangle$. In order to do this the following steps are required.

1. We define a function $R(\theta_k)$ as follows:

$$R(\theta_k) = \begin{cases} 0 & \text{if } 0 \leq \theta_k < \pi \\ 1 & \text{if } \pi \leq \theta_k < 2\pi \end{cases} \quad (2.8)$$

Therefore,

$$\exp(i\pi R(\theta_k)) = \begin{cases} 1 & \text{if } 0 \leq \theta_k < \pi \\ -1 & \text{if } \pi \leq \theta_k < 2\pi \end{cases} \quad (2.9)$$

The point of doing this is the variational need to optimize over angles in \mathbb{R} . The function R , shown in Figure 2.1, converts a continuous variable into a binary one, which is what we need.

2. Given a graph $G(V, E)$ such that $|V| = n$ and $|E| = m$, to create the ansatz we first define the number of qubits required as follows:

$$N = \lceil \log_2 n \rceil \quad (2.10)$$

When the number of nodes are not an exact power of 2, we can adjust L to be of size 2^N by adding null matrices of size $2^N - |V|$, $\mathbf{O}_{2^N - |V|}$, as shown in line 3, Algorithm 1.

3. Create a quantum circuit and apply a Hadamard gate to all the qubits to achieve an equal superposition of the states (lines 7 and 8, Algorithm 1).
 4. To the circuit, apply a diagonal gate U (line 9, Algorithm 1) of the following form:

$$U(\theta) = \begin{bmatrix} e^{i\pi R(\theta_1)} & 0 & 0 & \dots \\ 0 & e^{i\pi R(\theta_2)} & 0 & \dots \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & e^{i\pi R(\theta_n)} \end{bmatrix} \quad (2.11)$$

Therefore the final ansatz is:

$$|\Psi(\theta)\rangle = U(\theta)H^{\otimes N}|0\rangle^{\otimes N} \quad (2.12)$$

The state in the above equation is obtained in line 10 of Algorithm 1.

Having an ansatz, we can now define the Laplacian as an observable and evaluate the measurement (as in equation 2.7) which is the energy of the system. Since the classical optimizer minimizes the cost function we take the negative of the Laplacian matrix. Thus the final cost function is:

$$C(\theta) = -2^{N-2} \langle \Psi(\theta) | L | \Psi(\theta) \rangle \quad (2.13)$$

To evaluate this expectation value, the Laplacian matrix needs to be converted into a sum of tensor products of Pauli matrices (line 4 Algorithm 1). This is because in order to calculate the expectation value of L we need to express our quantum state in the L -basis. Converting to a general basis like L can be difficult, but converting to Pauli basis is relatively straightforward. For more details, see section 1.1.5).

Using classical black-box meta optimizers such as COBYLA, Nelder-Mead or Genetic Algorithm (as detailed in Algorithm 2), we then obtain

$$C^*(\theta^*) = \min C(\theta) \quad (2.14)$$

The final parameters obtained θ^* gives the bi-partition vector, using equation (2.9).

The entire process is expressed in the Figure 2.2. We start the algorithm from the top left corner box by creating a variational ansatz of N -qubits. In the top right corner box, this is followed the by calculation of expectation value of the Hamiltonian H which in our case is $-L$ or the negative of the Laplacian matrix (H in the top right corner box is the Hamiltonian and is not be confused with the Hadamard gate which is present in the variational ansatz). Then we move down to the classical part where we run the classical black-box optimizer loop in order to optimize the parameters. This optimization loop is shown in Algorithm 2. The parameters are updated and fed back to the variational ansatz until the best solution is found. At this point (bottom-most box), we take the final parameters and the value of the cost function as our results, calculate bi-partition vector from the values of the parameters.

2.2.1.2 Step-by-step explanation of LogQ for MAXIMUM CUT with an example

In this section we will give a step-by-step run-through of LogQ for the MAXIMUM CUT problem for a given graph. Let the given graph be $G(V, E, w)$ such that $V \in \{1, 2, 3, 4\}$, $E \in \{(1, 2), (1, 3), (2, 3), (3, 4)\}$ and $w \in \{3, 1, 8, 4\}$.

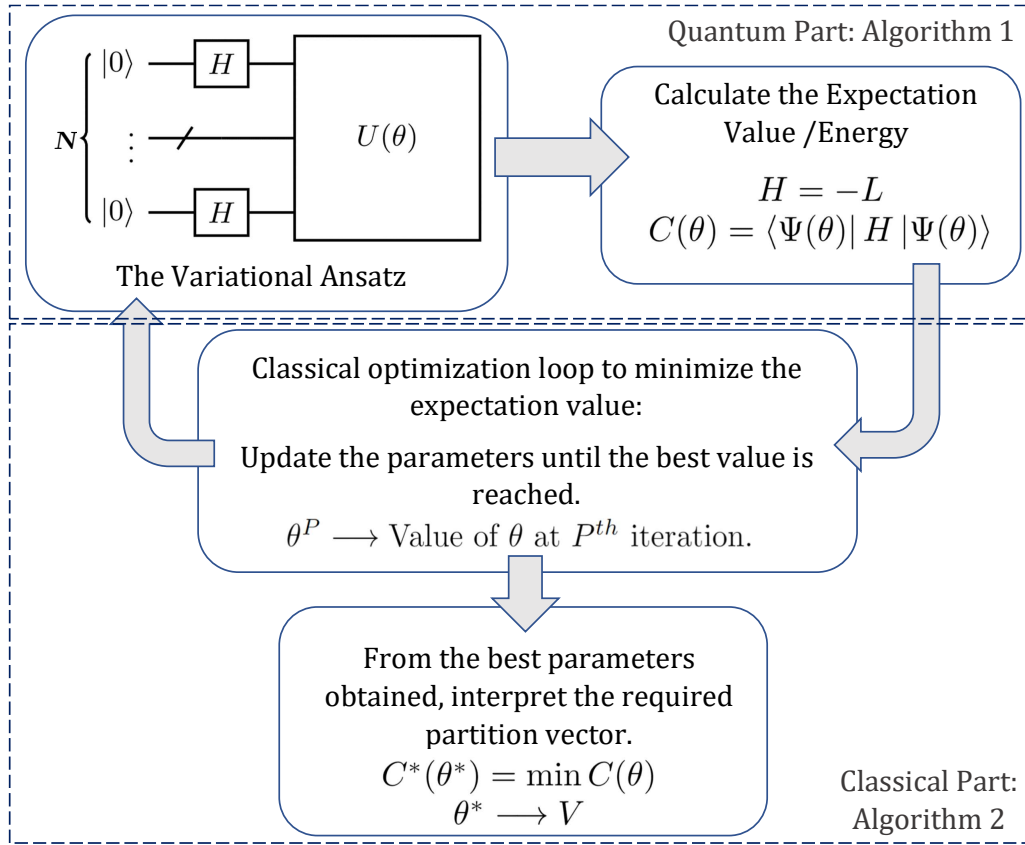


Figure 2.2: Diagrammatic representation of the hybrid quantum-classical algorithm using the LogQ encoding.

Algorithm 1: LogQ Encoding of MAXIMUM CUT: Building the Objective Function

Input: Laplacian matrix of a graph $G(V, E)$

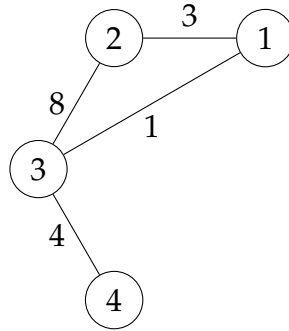
- 1 $L \leftarrow$ Laplacian matrix of size $|V| \times |V|$
- 2 $N \leftarrow \lceil \log_2 |V| \rceil$
- 3 $L^* \leftarrow \begin{bmatrix} L & \mathbf{O}_{2^{N-|V|}} \\ \mathbf{O}_{2^{N-|V|}} & \mathbf{O}_{2^{N-|V|}} \end{bmatrix}$
- 4 $H \leftarrow \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot L^*) J_i$ where $J = \{\prod_{k=1}^N S^{\otimes k}\}$
- 5 $\theta \leftarrow$ List of $|V|$ parameters
- 6 **Function** EvalCost(θ):
- 7 $Q \leftarrow$ Quantum Circuit of N qubits
- 8 Add Hadamard gate to each Qubit
- 9 $U \leftarrow$ diagonal gate $diag(\theta, R)$
- 10 Apply U to Q
- 11 $F \leftarrow$ ExpectationValue(Q, H)
- 12 **return** $2^{|V|-2} F$
- 13

Algorithm 2: LogQ Encoding of MAXIMUM CUT: Minimizing the Objective Function

Input: EvalCost(θ)

```

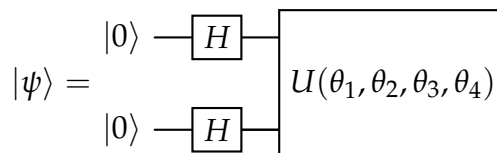
1 Function Optimizer(EvalCost( $\theta$ ),  $\theta^{initial}$ ):
2   repeat
3      $\theta^p \leftarrow \theta$  at  $p^{th}$  iteration
4      $C \leftarrow$  EvalCost( $\theta^p$ )
5     if  $C$  is sufficiently good then
6        $C^* \leftarrow C$ 
7       break
8     else
9       Update  $\theta^p \rightarrow \theta^{p+1}$ 
10      continue
11    end
12  return  $C^*$ 
13
```



The Laplacian matrix of this graph is:

$$L = \begin{bmatrix} 4 & -3 & -1 & 0 \\ -3 & 11 & -8 & 0 \\ -1 & -8 & 13 & -4 \\ 0 & 0 & -4 & 4 \end{bmatrix} \quad (2.15)$$

Since the size of the problem is $n = 4$, we need $\log_2 4 = 2$ qubits. We create the initial quantum circuit with 2 qubits as follows:



The circuit above consists of 2 Hadamard gates and then a diagonal gate U . This diagonal gate implements the following matrix as gate:

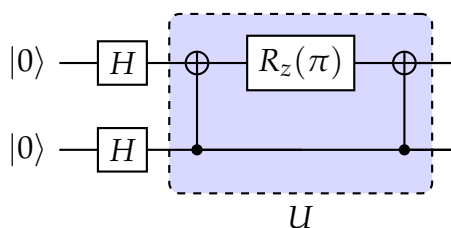
$$U = \begin{bmatrix} e^{i\pi R(\theta_1)} & 0 & 0 & 0 \\ 0 & e^{i\pi R(\theta_2)} & 0 & 0 \\ 0 & 0 & e^{i\pi R(\theta_3)} & 0 \\ 0 & 0 & 0 & e^{i\pi R(\theta_4)} \end{bmatrix} \quad (2.16)$$

This diagonal gate can be easily implemented in the *qiskit* Python package using the *diag* function. The R , as explained before, is a simple function that converts a continuous variable to a binary 0 or 1. Therefore we can consider $R(\theta_i)$ as a binary variable.

The exact gate to be implemented depends on the values of θ_i and hence $R(\theta_i)$. Let $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$. Let us take for example the case where $\theta = \{0.5\pi, 1.4\pi, 1.7\pi, 0.8\pi\}$, such that: $R(\theta) = \{0, 1, 1, 0\}$. Then the gate to be implemented is:

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.17)$$

For this particular case the circuit becomes:



It can be verified that the above circuit does represent the diagonal gate U . This can be done by multiplying the three matrices representing the gate. The first gate is the CNOT followed by the R_z gate on the first qubit (this can be represented over two qubits a $R_z \otimes I$ since there is nothing on the second qubit). Finally there is another CNOT. Note that the CNOT here will be different from the CNOT gate shown in table 1.2. This is because here, unlike in table 1.2, the first qubit is the target and the second qubit is the control.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} e^{-i\pi/2} & 0 \\ 0 & e^{i\pi/2} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.18)$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.19)$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} -i & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & i \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.20)$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} -i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \quad (2.21)$$

$$= \begin{pmatrix} -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \end{pmatrix} \quad (2.22)$$

$$= -i \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.23)$$

Which is exactly the same as the expression of U in equation [2.17](#) upto a global phase $-i$. Note that a global phase does not change the resulting quantum state.

After constructing the above circuit, we need to calculate the expectation value of the observable (defined in [\(2.24\)](#)), which in our case is the Laplacian matrix. To do this we need to first convert the Laplacian matrix into a sum of Pauli strings. After converting the Laplacian matrix, we get the following:

$$L = 8II - 3.5IX + 0.5IZ - 0.5XI - 4XX - 0.5XZ - 4YY - 0.5ZI + 0.5ZX - 4ZZ \quad (2.24)$$

Here $IX = I \otimes X$, $IZ = I \otimes Z$ and so on are the tensor products between Pauli matrices.

To see that the Laplacian can be represented as in equation [\(2.24\)](#), we can compute the expression as follows:

$$\begin{aligned}
 L = & 8 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 3.5 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\
 & - 0.5 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 4.0 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} - 0.5 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\
 & - 4.0 \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} - 0.5 \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\
 & - 4.0 \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.25}
 \end{aligned}$$

$$\begin{aligned}
 = & 8 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - 3.5 \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} - 0.5 \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \\
 & - 4 \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} - 0.5 \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} - 4 \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \\
 & - 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} + 0.5 \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix} - 4 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.26}
 \end{aligned}$$

$$= \begin{pmatrix} 8 + 0.5 - 0.5 - 4 & -3.5 + 0.5 & -0.5 - 0.5 & -4 + 4 \\ -3.5 + 0.5 & 8 - 0.5 - 0.5 + 4 & -4 - 4 & -0.5 + 0.5 \\ -0.5 - 0.5 & -4 - 4 & 8 + 0.5 + 0.5 + 4 & -3.5 - 0.5 \\ -4 + 4 & -0.5 + 0.5 & -3.5 - 0.5 & 8 - 0.5 + 0.5 - 4 \end{pmatrix} \tag{2.27}$$

$$= \begin{pmatrix} 4 & -3 & -1 & 0 \\ -3 & 11 & -8 & 0 \\ -1 & -8 & 13 & -4 \\ 0 & 0 & -4 & 4 \end{pmatrix} \tag{2.28}$$

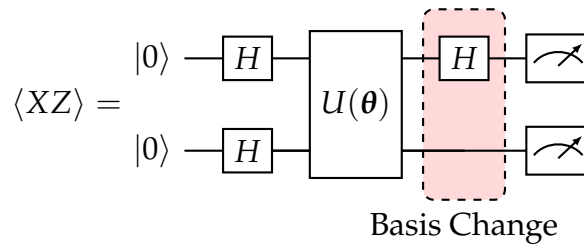
Therefore the expression gives the Laplacian matrix in equation (2.15).

The expectation value of the L , $\langle L \rangle$, is broken down into the expectation value of the terms.

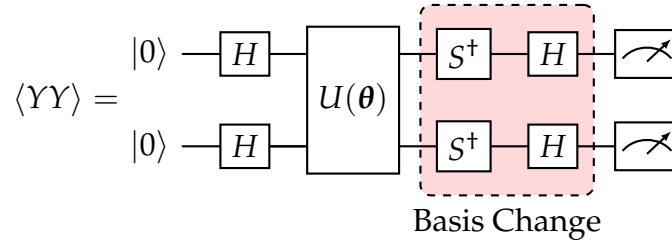
$$\begin{aligned}
 \langle L \rangle = & 8 \langle II \rangle - 3.5 \langle IX \rangle + 0.5 \langle IZ \rangle - 0.5 \langle XI \rangle - 4 \langle XX \rangle - 0.5 \langle XZ \rangle - 4 \langle YY \rangle - 0.5 \langle ZI \rangle \\
 & + 0.5 \langle ZX \rangle - 4 \langle ZZ \rangle \tag{2.29}
 \end{aligned}$$

There are 10 expectation values in this equation and for each expectation we need to measure a different circuit. To do this we will need to use the Table 1.3 to see which gate we need for each Pauli term. This is done for the change of basis, as explained in section 1.1.5. For the term XX we use the H on both qubits, for XZ , H on the first qubit and nothing (I) on the second qubit, and so on.

To calculate $\langle XZ \rangle$ we have this circuit:



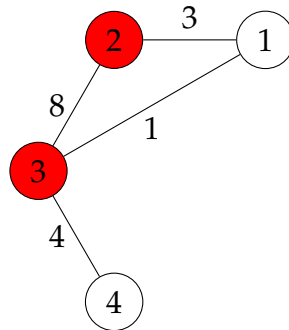
and for the $\langle YY \rangle$ we have:



and so on.

There will be 10 such circuits, one for every term. We can use equation (1.38) to calculate the expectation value of each term. Hence we can calculate $\langle L \rangle$ by substituting the expectation values of all the terms in equation 2.29. To get the actual value of the cost function we need to finally multiply this by the normalization factor $2^{N-2} = 2^{2-2} = 1$ ($N = 2$ is the number of qubits).

Let us go back to the example we considered before with $\theta = \{0.5\pi, 1.4\pi, 1.7\pi, 0.8\pi\}$ and hence $R(\theta) = \{0, 1, 1, 0\}$. This represents the bi-partition $\{1, 4\}$ and $\{2, 3\}$.



It is easy to see that the cost function here is 8 (3+1+4).

Let us calculate the expectation values of the different terms for this example. The circuit and results for the XX term are shown in figures 2.3 and 2.4 respectively. The circuit and the corresponding results are taken from the IBM Quantum platform.

Important note: The order of the qubits are inverted in the IBM platform visualization. The states on the X-axis are actually in the order 00, 10, 01 and 11.

For two qubits the formula to calculate the expectation value from the probability distribution is $p(00) - p(01) - p(10) + p(11)$ (explained in section 1.1.5). $p(00)$ is the probability of 00 and so on.

So, from 2.4 we can calculate $\langle XX \rangle$ as $0 - 0 - 0 + 1 = 1$.

Let us now calculate the term $\langle IX \rangle$. In this case we will only measure the second qubit. The circuit and probability distribution are shown in figures 2.5 and 2.6 respectively. In this case, since we only measured the second qubit we will calculate using the formula $p(0) - p(1)$ taking only the second qubit into account.

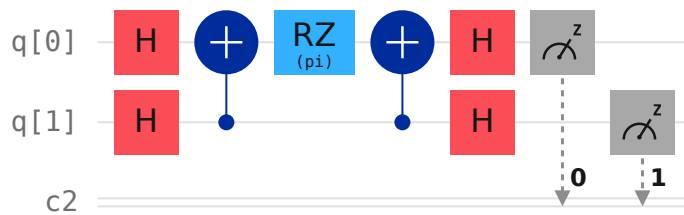


Figure 2.3: Circuit to calculate $\langle XX \rangle$

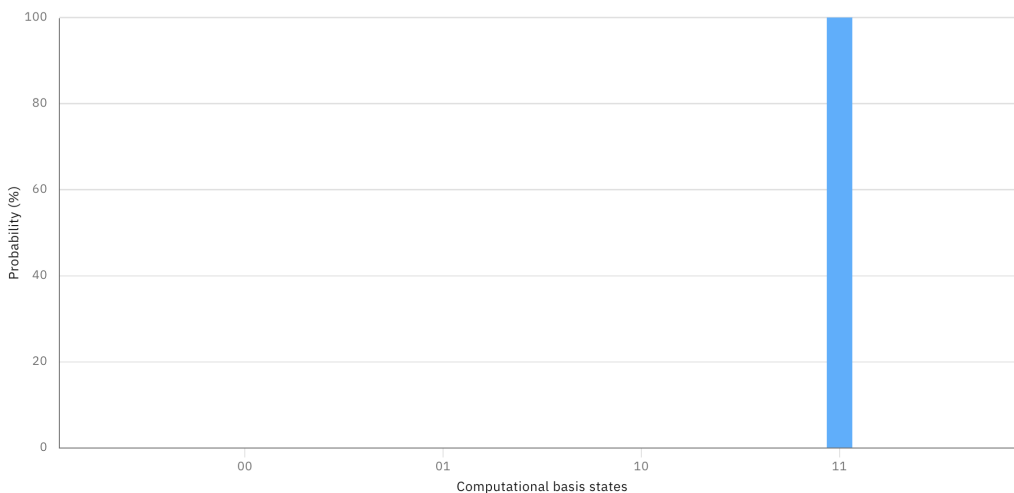


Figure 2.4: Probability distribution for $\langle XX \rangle$.

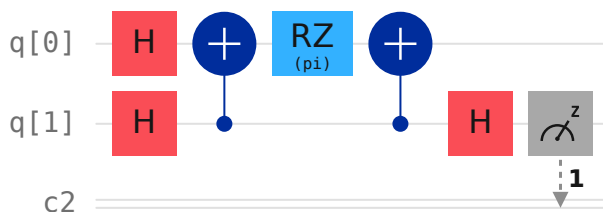


Figure 2.5: Circuit to calculate $\langle IX \rangle$

In figure 2.6, as mentioned before, the qubit labels are inverted so in fact the the state $|01\rangle$ has a probability of 1. We measured only the second qubit and therefore $p(1) = 1$. Similarly, $p(0) = 0$ and therefore $\langle IX \rangle = p(0) - p(1) = -1$.

The term $\langle II \rangle$ is trivial and is always 1 since $\langle \psi | II | \psi \rangle = \langle \psi | \psi \rangle = 1$.

Similarly all the other terms are calculated as follows:

- $\langle IZ \rangle = 0.5 - 0.5 = 0$
- $\langle XI \rangle = 0 - 1 = -1$
- $\langle XZ \rangle = 0 - 0.5 - 0 + 0.5 = 0$
- $\langle YY \rangle = 0.25 - 0.25 - 0.25 + 0.25 = 0$

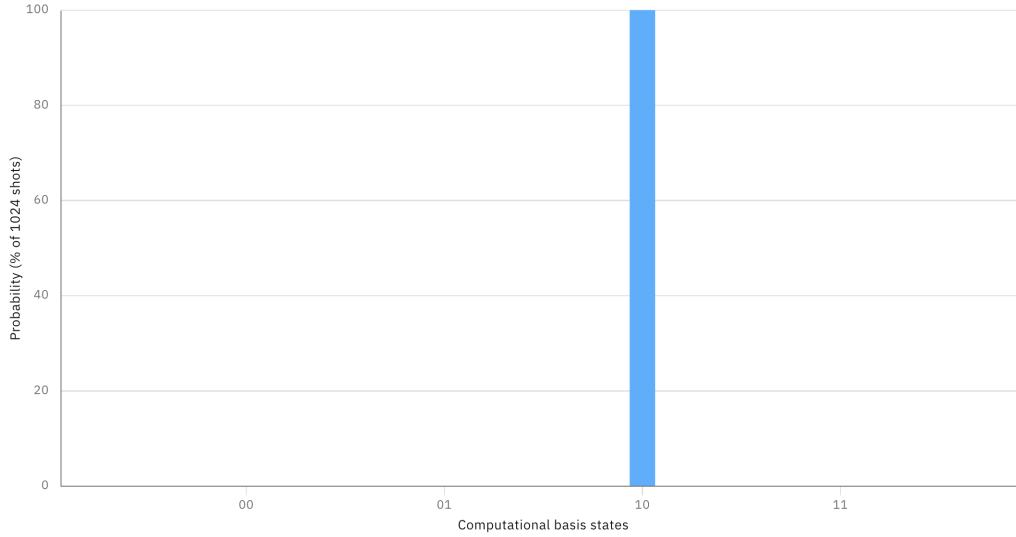


Figure 2.6: Probability distribution for $\langle IX \rangle$.

- $\langle ZI \rangle = 0.5 - 0.50 = 0$
- $\langle ZX \rangle = 0 - 0 + 0.5 - 0.5 = 0$
- $\langle ZZ \rangle = 0.25 - 0.25 - 0.25 + 0.25 = 0$

Substituting these values in the equation (2.29), we have:

$$\begin{aligned}
 \langle L \rangle &= 8 \times 1 - 3.5 \times (-1) + 0.5 \times 0 - 0.5 \times (-1) - 4 \times 1 - 0.5 \times 0 - 4 \times 0 - 0.5 \times 0 \\
 &\quad + 0.5 \times 0 - 4 \times 1 \\
 &= 8 + 3.5 + 0.5 - 4 \\
 &= 8
 \end{aligned} \tag{2.30}$$

Finally multiply the normalization factor $2^{N-2} = 1$. So $8 \times 1 = 8$. This is exactly the cost function expected for the parameters $\theta = \{0.5\pi, 1.4\pi, 1.7\pi, 0.8\pi\}$.

This concludes the quantum part of the algorithm. We have a full quantum circuit that depends on 4 parameters and generates the value of the cost function. Now we use a classical black box optimizer in order to optimize these parameters so that we get the best value of the cost function. The input to the black box optimizer are the 4 values of θ , all initialized to 0, and the objective function which is a Python function that creates the quantum circuit and outputs the cost function as shown above.

After having optimized the parameters, we get back the set of 4 parameters. Figure 2.7 shows the screenshot of the results after the parameters are optimized for the instance in this example. The optimized parameters are $\theta = \{4.85, 2.69, 4.87, 2.19\}$ which gives $R(\theta) = \{1, 0, 1, 0\}$ (0 when less than $\pi = 3.14$).

This means that the solution to our problem is the bi-partition $\{1, 3\}$ and $\{2, 4\}$. It can be verified that the value of the cost function for this bi-partition is 15, which is also the value of the cost function given by the Genetic Algorithm in Figure 2.7 (the value shown is -15 but the negative sign is because we minimize the expectation value of $-L$ which is equivalent to maximizing the expectation value of L).

```
The best solution found:
[4.84667416 2.68955801 4.86832798 2.19270913]

Objective function:
-15.0
```

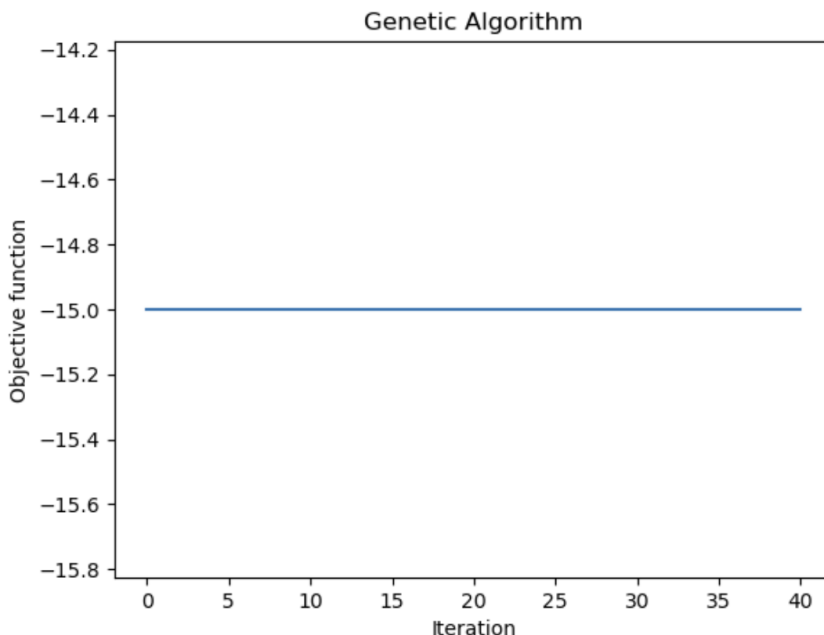


Figure 2.7: Result after run on black-box optimizer, Genetic Algorithm in this case.

2.2.1.3 Complexity analysis of LogQ

LogQ helps us represent large problems (by current standards of quantum computing) on a quantum computer. Algorithms like the QAOA, for example, require 128 qubits to represent a 128-node MAXIMUM CUT problem. The same problem can be solved by LogQ using only 7 qubits. It therefore has the promise of being able to be applied to interesting and even industrially relevant sizes using the currently available sizes of NISQ QCs.

Here we will try to analyse the time complexity of LogQ with respect to depth and density of the problem.

The number of CNOT gates required for the QAOA ansatz is $p|E|$, where p is the depth of the algorithm and $|E|$ is the number of edges in the graph. In the worse-case scenario, or when the graph is a clique, $|E| = |V|^2$.

The time taken to do a complete measurement will depend the number of Pauli terms in the decomposition of the Hamiltonian matrix. This is because to evaluate every Pauli term we require a *separate* circuit (see equation (1.51)). For a graph of size $|V|$, we will need $N = \lceil \log_2 |V| \rceil$ qubits. In the worst-case scenario we have 4^N Pauli terms to measure. Therefore, the complexity for a measurement will be $O(4^N)$. An increase in density of the graph will increase the number of Pauli terms. Since $N = \lceil \log_2 |V| \rceil$, the complexity is therefore $O(4^{\lceil \log_2 |V| \rceil}) = O(|V|^2)$.

Note that this is exactly the same complexity as if the expectation were calculated classically. For the vertex whose corresponding θ value is between $[0, \pi]$, it is put into the first partition. For vertex whose corresponding θ values are between $[\pi, 2\pi]$, it is put into the second partition. Then the expectation can be calculated if one considers whether all the edges are cut or not. The process can be finished within $O(V^2)$.

However, for sparse matrices, much more efficient schemes to decompose the Hamilto-

nian into m -summands and calculate the expectation values with less than m -queries to the quantum computer exist (where $m \ll |V|^2$). On the decomposition side, an efficient way to decompose sparse Hermitian matrices to sums of unitaries exist. This is described in detail in Appendix A of Ref. [62]. Furthermore, an expectation value of a sum of m summands can be obtained in less than m queries to the QPU, again for sparse operators. More details can be seen in Ref. [63].

In order of calculate one expectation value, we need to evaluate $O(|V|^2)$ circuits. In a single circuit the number of CNOTs is equal to $|V| - 1$, $|V|$ being the number of vertices. Hence we have a total of $|V|^3$ CNOTs.

While QAOA has $p|E| \leq p(|V|^2 - |V|) / 2$ number of CNOT gates, all our circuits are of depth $|V|$ which makes QAOA dramatically more sensitive to errors (because it has a quadratically more CNOT gates). Moreover, for all practical purposes, $p \gg 1$ [64], and hence $\propto p|V|^2$ CNOTs could be comparable or even greater than our total of $|V|^3$ CNOTs .

Finally, in our study to propose a new encoding compatible with NISQ, the search space remains the same as that of the classical search space. A procedure to reduce the number of variables has been presented in [57]. However, this method is beyond the scope of the current work. Readers should also refer to followup studies [65] where the algorithm was evaluated on the MAXIMUM CUT problem by using the alternating optimization procedure [66] which scales polynomially in problem size. The analysis of this section is summarized in Table 2.1.

Property	LogQ	QAOA	Classical
Solutions handled simultaneously	1	$2^{ V }$	1
No. of Qubits	$\lceil \log_2 V \rceil$	$ V $	-
No. of CNOTs	$ V ^3$	$p(V ^2 - V)$	-
Number of measurements	$ V ^2$	1	-
Circuit depth per measurement	$ V $	$p(V ^2 - V)$	-

Table 2.1: Comparison between LogQ, QAOA and Classical Algorithm.

2.2.2 Applying the algorithm to other NP-hard problems

A logical next step is to attempt to solve a variety of combinatorial optimisation problems using the algorithm. In Karp's paper from 1972 [26], he outlined how we can convert one NP-complete problem into another. A more recent paper [27] lists numerous more such reductions. Figure 2.8 shows a subset (a transformation family) of these reductions directly or indirectly relating to MAXIMUM CUT. Here, we follow a similar logic to convert various NP-hard problems to MAXIMUM CUT.

Note, however, that these conversions might not have a one-to-one scaling. For example, an n variable MAXIMUM 2-SAT problem requires us to solve a $2n$ node MAXIMUM CUT problem.

In Karp's paper all the transitions are from one decision problem to another. Usually in classical computing it would be considered trivial to convert a decision problem into an optimisation one. However, our algorithm is inherently an optimisation algorithm and moreover will give various results for a various runs. The point being, it will not respond well to yes-no decision problems. Therefore, it is important to make reductions between the optimisation versions. Moreover it is important to make sure that these conversions support a wide definition of the problems (for example MAXIMUM 3-SAT instead of 3-SAT).

Following are some such polynomial-time reductions of NP-hard problems:

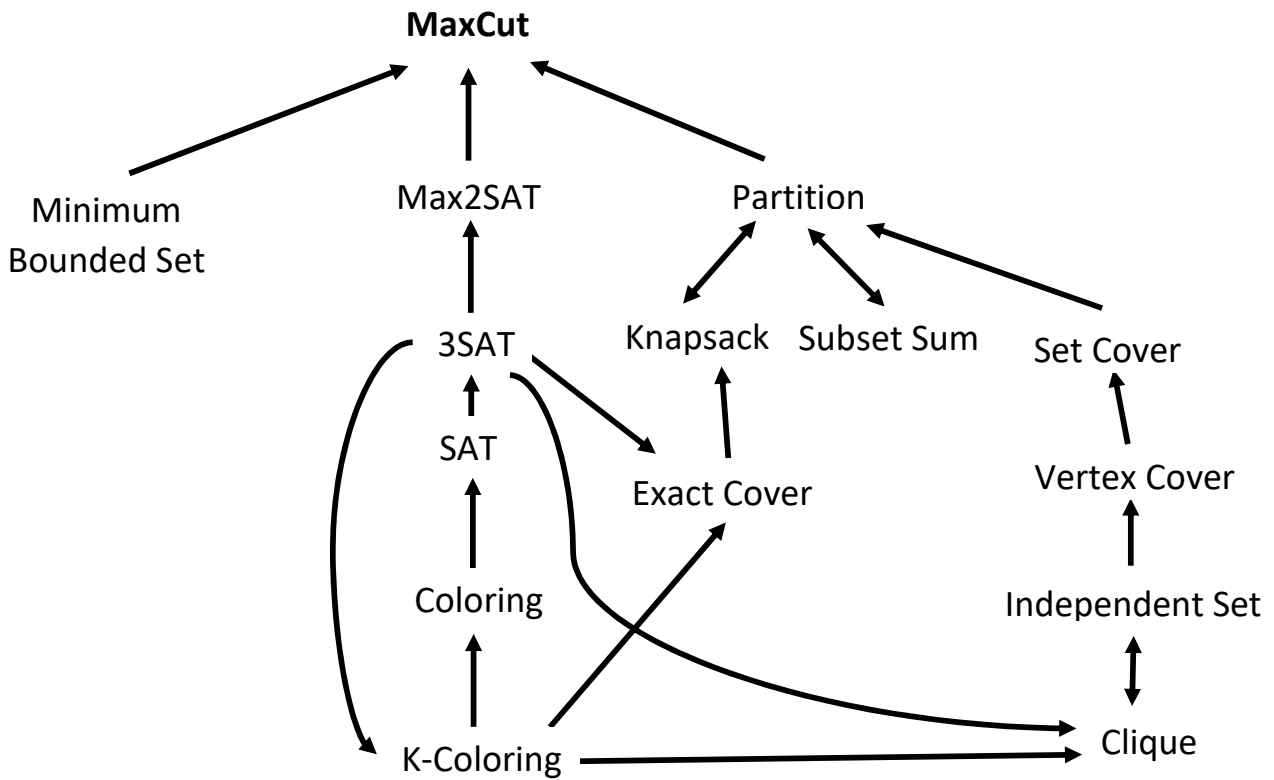


Figure 2.8: Graph of MAXIMUM CUT transformation family for NP-complete problems.

2.2.2.1 MINIMUM PARTITION to MAXIMUM CUT

MINIMUM PARTITION

Input A set $S = \{w : w \in \mathbb{Z}^+\}$.

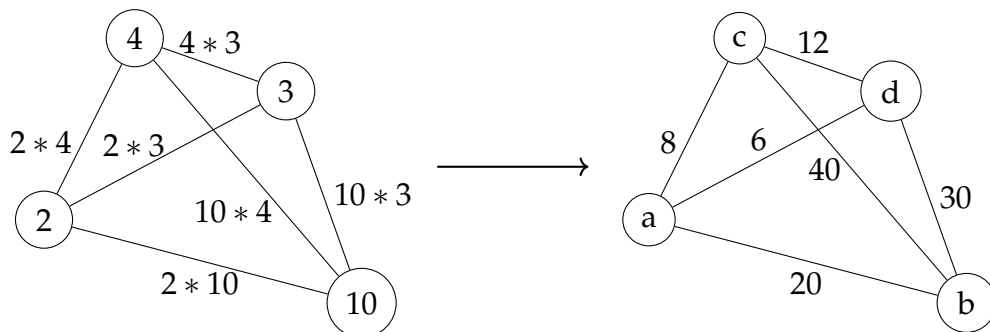
Task Find $A \subseteq S$ that minimizes

$$\left| \sum_{w_k \in A} w_k - \sum_{w_l \notin A} w_l \right|.$$

This can be converted to the maximum cut problem in the following manner:

1. Create a graph such that there is a node for every number.
2. For every pair of nodes (i, j) , connect them using an edge of weight $w_i * w_j$.
3. The maximum cut value of this graph gives a bi-partition that is equivalent to the minimum partition.

For example, if the input set S of the MINIMUM PARTITION problem is $\{2, 10, 4, 3\}$ then it is equivalent to the MAXIMUM CUT problem on the following graph:



The solution to the MAXIMUM CUT instance above is $\{a, c, d\}$ and $\{b\}$ which gives the value of the cost function as 90. This solution in turn gives the solution to the MINIMUM PARTITION instance, which is $\{2, 4, 3\}$ and $\{10\}$. The nodes $\{a, c, d\}$ are equivalent to the numbers $\{2, 4, 3\}$ as can be seen in the figure above. The value of the cost function is $1(10 - (2 + 4 + 3))$.

This conversion is $n \rightarrow n$ in terms of number of variables and $n \rightarrow n^2$ in terms of data.

2.2.2.2 MAXIMUM 2-SAT to MAXIMUM CUT

MAXIMUM 2-SAT

Input A set of m clauses $C = \{w_{pq}(x_p + x_q) : x_p, x_q \in X \cup X'\}$ where $X = \{x : x \in \{0, 1\}\}$, $X' = \{\bar{x} : x \in X\}$ and w_{pq} are the clause weights.

Task Find the variable assignment X that maximizes the combined weight of the satisfied clauses.

The problem is said to be satisfiable if all the clauses are satisfied.

We can convert this problem into the maximum cut problem in the following manner:

1. In a graph, assign 2 nodes for every variable, one for the variable and another for the complement of the variable. Hence there are $2|X|$ nodes in the graph.
2. Draw an edge between the nodes representing the variables and their complements. For example connect x_1 and \bar{x}_1 , x_2 and \bar{x}_2 and so on. Add a large edge weight equal to the sum all clause weights. This is to make sure that the variables and their complements do not fall in the same partition.
3. For every clause, add an edge between the respective nodes with edge weight as w_{pq} .
4. The maximum cut of this graph is equivalent to the MAXIMUM 2-SAT solution.

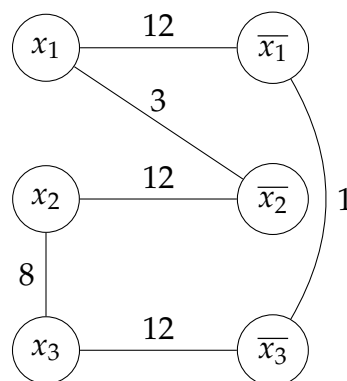
Consider a 3-variable MAXIMUM 2-SAT problem with the following clauses and associated weights:

1. $x_1 + \bar{x}_2, 3$

2. $\bar{x}_1 + \bar{x}_3, 1$

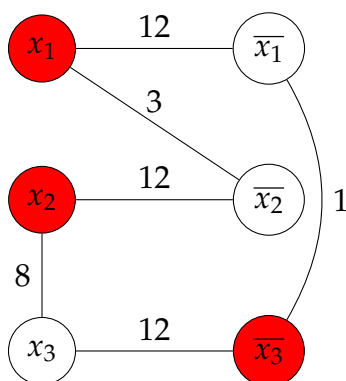
3. $x_2 + x_3, 8$

In order to convert it to the MAXIMUM CUT problem, we make the following graph:



The sum of the clause weights is 12. The weights of the edges between the variables and their complements are therefore set to 12. Then there are edges between nodes according to the clauses.

The solution to this MAXIMUM CUT instance is (in red):



Here the value of the MAXIMUM CUT cost function is 48. From this we get the MAXIMUM 2-SAT solution as $x_1 = 1, x_2 = 1, x_3 = 0$, which satisfies all the clauses and hence the value of the cost function is the sum of the weights $8 + 3 + 1 = 12$.

This conversion is $n \rightarrow 2n$ in terms of number of variables. A n variable MAXIMUM 2-SAT translates to a $2n$ node MAXIMUM CUT. In terms of data this conversion is $3m \rightarrow m + n$ (n being the number of variables and m the number of clauses).

2.2.2.3 MAXIMUM CLIQUE to MAXIMUM 2-SAT

MAXIMUM CLIQUE

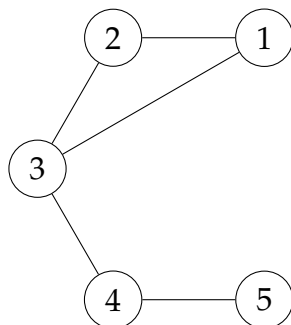
Input A graph $G(V, E)$.

Task Maximize $|V'|$ in the graph $\{G'(V', E') : V' \subseteq V, E' \subseteq E, |E'| = \frac{|V'|(|V'|-1)}{2}\}$.

It can be converted to the MAXIMUM 2-SAT problem in the following manner:

1. Consider a graph $G(V, E)$ having vertices $v_i \in V$. For each vertex v_i add a variable x_i . Also an auxiliary variable z . We therefore have $|V| + 1$ variables.
2. For every variable add the following 2 clauses: $(x_i + z)$ and $(x_i + \bar{z})$. Let us refer to these clauses as Type A clauses.
3. Add the following clauses: $(\bar{x}_i + \bar{x}_j) \forall (i, j) \notin E$. We will refer to these clauses as clauses of type B.
4. The clauses of type A ensure that the maximum number of nodes are selected and the clauses of type B make sure that the selected subgraph is a clique.
5. To the type B clauses, add a large weight. Due to the nature of the algorithm and it's susceptibility to errors, we may get solutions that are not cliques at all. Moreover finding a clique and maximizing it are 2 different problems and by adding weights we make sure that they are not affected by one another.
6. The MAXIMUM 2-SAT problem is solved for this set of clauses. The partition of selected variables form the MAXIMUM CLIQUE.

Consider the following input graph for the MAXIMUM CLIQUE problem:



In order to convert it to the MAXIMUM 2-SAT problem we will need 6 variables of which 1 is an auxillary variable. The equivalent MAXIMUM 2-SAT problem will have the following clauses:

1. Type A clauses and respective weights:

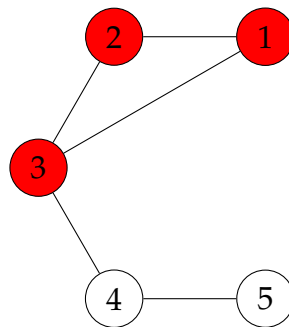
- | | | |
|------------------|------------------------|------------------------|
| (a) $x_1 + z, 1$ | (e) $x_5 + z, 1$ | (i) $x_4 + \bar{z}, 1$ |
| (b) $x_2 + z, 1$ | (f) $x_1 + \bar{z}, 1$ | (j) $x_5 + \bar{z}, 1$ |
| (c) $x_3 + z, 1$ | (g) $x_2 + \bar{z}, 1$ | |
| (d) $x_4 + z, 1$ | (h) $x_3 + \bar{z}, 1$ | |

2. Type B clauses and respective weights:

- | | | |
|---------------------------------|---------------------------------|---------------------------------|
| (a) $\bar{x}_1 + \bar{x}_4, 10$ | (c) $\bar{x}_2 + \bar{x}_4, 10$ | (e) $\bar{x}_3 + \bar{x}_5, 10$ |
| (b) $\bar{x}_1 + \bar{x}_5, 10$ | (d) $\bar{x}_2 + \bar{x}_5, 10$ | |

Here we have chosen the Type B clause weights to be the sum of all the Type A clause weights. This is done to make sure that the Type B clauses are prioritized to generate a feasible solution.

The solution the MAXIMUM 2-SAT problem above is $x_1 = x_2 = x_3 = z = 1, x_4 = x_5 = 0$. This solution satisfies all type B clauses and 8 type A clauses. This solution is also the solution to the MAXIMUM CLIQUE problem (in red):



This conversion is $n \rightarrow n + 1$ in terms of number of variables. A n variable MAXIMUM CLIQUE translates to a $n + 1$ variable MAXIMUM 2-SAT. In terms of data this conversion is $n^2 \rightarrow 3m$ (n being the number of nodes of the graph and m the number of clauses of the MAXIMUM 2-SAT problem).

Note that in order to write a MAXIMUM CLIQUE problem as a MAXIMUM CUT, one would need to undergo 2 conversions. In that case a n variable MAXIMUM CLIQUE translates to a $2(n + 1)$ variable MAXIMUM CUT.

2.2.3 Generalizing the LogQ Encoding

The LogQ encoding solves, originally, the MAXIMUM CUT problem. Various conversions are then used in order to solve other problems. Here, a second, more general approach, shall be described, where any problem which can be written in the form of a Quadratic Unconstrained Binary Optimization (QUBO) problem [31] can be solved. Instead of taking the Laplacian matrix as the input, this algorithm takes as input the QUBO matrix of the problem.

Firstly, we define the QUBO matrix. To describe a problem as a QUBO, all the terms in the objective function should be either linear or quadratic. Since the variables in the objective function are binary, a linear term can be easily converted to a quadratic one, since $x_i^2 = x_i \forall x \in \{0, 1\}$.

Consider the objective function of the following form:

$$P = \sum_{ij} a_{ij} x_i x_j \quad (2.31)$$

It can be rewritten as:

$$P = (x_1 \ x_2 \ \dots \ x_n) \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \quad (2.32)$$

$$P = x^T Q x \quad (2.33)$$

$$Q = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (2.34)$$

Q is the required QUBO matrix.

This matrix cannot be directly used in the algorithm. This is because in the original MAXIMUM CUT algorithm, the variable used belongs to the set $\{1, -1\}$ and not $\{0, 1\}$. To make the equation mathematically consistent, we need to reformulate the QUBO matrix.

Let $z \in \{1, -1\}$ and $x \in \{0, 1\}$, then $x = \frac{1-z}{2}$. Equation (2.31) therefore becomes:

$$P = \sum_i a_{ii} \frac{1-z_i}{2} + \sum_{\substack{ij \\ i \neq j}} a_{ij} \frac{1-z_i}{2} \frac{1-z_j}{2} \quad (2.35)$$

Note that in the first term $\frac{1-z_i}{2}$ has been used instead of $\left(\frac{1-z_i}{2}\right)^2$ since $x_i^2 = x_i$.

In the search for optimal values of parameters z we can eliminate the constant terms in P as they only add a constant shift to the cost function. We can therefore simplify (2.35) as follows:

$$P = \sum_i a_{ii} \frac{1-z_i}{2} + \sum_{\substack{ij \\ i \neq j}} a_{ij} \frac{1-z_i}{2} \frac{1-z_j}{2} \quad (2.36)$$

$$= \frac{1}{2} \sum_i a_{ii} (1-z_i) + \frac{1}{4} \sum_{\substack{ij \\ i \neq j}} a_{ij} (1-z_i - z_j + z_i z_j) \quad (2.37)$$

$$= -\frac{1}{2} \sum_i a_{ii} z_i + \frac{1}{4} \sum_{\substack{ij \\ i \neq j}} a_{ij} (-z_i - z_j + z_i z_j) \quad (2.38)$$

The above *cannot* be represented in a matrix form similar to Eq. (2.34) since it has linear terms that cannot be quadratized since $z_i^2 \neq z_i$.

We therefore need to reformulate the problem. In this reformulation the linear terms are represented in the off-diagonal terms instead of the diagonals. Let us have $2n$ variables $\{z_1, z_2, \dots, z_{2n}\}$ where $z_1 \dots z_n \in \{1, -1\}$ and $z_{n+1} \dots z_{2n} \in \{1\}$. Then to represent a linear variable z_i , we can have the term $z_i z_{i+n}$ where $z_{i+n} = 1$. The equation (2.38) can be rewritten as follows:

$$P = -\frac{1}{2} \sum_i a_{ii} z_i z_{i+n} + \frac{1}{4} \sum_{\substack{ij \\ i \neq j}} a_{ij} (-z_i z_{i+n} - z_j z_{j+n} + z_i z_j) \quad (2.39)$$

This is our reformulated QUBO, which we shall call the *spin-QUBO* (sQUBO). This is a matrix of size $2n \times 2n$. It will require $\lceil \log_2 2n \rceil = (1 + \lceil \log_2 n \rceil)$ qubits, or, in other words, 1 more qubit than the original algorithm. Note that this is still an optimization problem of n variables since the variables $z_{n+1} \dots z_{2n}$ are fixed.

Given a QUBO matrix Q of size n , the sQUBO matrix will have the following structure:

$$P = \begin{pmatrix} Q & D_n \\ D_n & \mathbf{0}_n \end{pmatrix} \quad (2.40)$$

Here D_n is a diagonal matrix of size n and $\mathbf{0}_n$ is a zero matrix of size n .

While the increase in the size of the matrix will increase the number of Pauli terms in the decomposition, converting the QUBO into sQUBO does not affect the complexity significantly. In this case there are $4^N = 4^{\log_2 2n} = 2^{2 \log_2 2n} = 2^{\log_2 4n^2} = 4n^2 = O(n^2)$ expectation values.

Using our formulation, we propose that any problem that can be represented in a QUBO format can be solved using the algorithm described in Algorithm 3.

Algorithm 3: LogQ Encoding of a QUBO problem: Building the Objective Function

Input: QUBO Matrix

- 1 Convert QUBO to sQUBO
 - 2 $Q \leftarrow$ sQUBO
 - 3 $N \leftarrow \lceil \log_2 2n \rceil$
 - 4 $Q^* \leftarrow \begin{bmatrix} Q & \mathbf{0}_{2^{N-n}} \\ \mathbf{0}_{2^{N-n}} & \mathbf{0}_{2^{N-n}} \end{bmatrix}$
 - 5 $H \leftarrow \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot Q^*) J_i$ where $J = \{\prod_{k=1}^N S^{\otimes k}\}$
 - 6 $\theta \leftarrow$ List of n parameters
 - 7 **Function** EvalCost(θ):
 - 8 QC \leftarrow Quantum Circuit of N qubits
 - 9 Add Hadamard gate to each Qubit
 - 10 $U \leftarrow$ diagonal gate $\text{diag}(\theta, R)$
 - 11 Apply U to QC
 - 12 $F \leftarrow$ ExpectationValue(Q, H)
 - 13 **return** F
 - 14
-

2.2.3.1 MAXIMUM WEIGHTED INDEPENDENT SET using QUBO

MAXIMUM WEIGHTED INDEPENDENT SET

Input A graph $G(V, E)$ with node weights w_i

Task Find $x \in \{0, 1\}^{|V|}$ that maximizes $\sum_i w_i x_i$ such that $x_i + x_j \leq 1 \forall (i, j) \in E$.

The MAXIMUM WEIGHTED INDEPENDENT SET problem consists of an objective function and constraints. We can however incorporate the constraints in the objective function as penalty terms. Let p be the magnitude of the penalty.

$$W = \max \left(\sum_i w_i x_i - p \left(\sum_{(i,j) \in E} x_i x_j \right) \right) \quad (2.41)$$

Since x_i is binary,

$$W = \max \left(\sum_i w_i x_i^2 - p \left(\sum_{(i,j) \in E} x_i x_j \right) \right) \quad (2.42)$$

Hence we have a QUBO matrix of the following form:

$$Q_{ij} = \begin{cases} w_i & \text{if } (i, j) \in E \text{ and } i = j \\ -\frac{p}{2} & \text{if } (i, j) \in E \text{ and } i \neq j \\ 0 & \text{if } (i, j) \notin E \end{cases} \quad (2.43)$$

Q_{ij} can now be used as input in Algorithm 3 to solve the problem.

2.2.4 Classical Models for Benchmarking

In order to assess the performance of our algorithm, we compare it with the results of classical optimization models. Following are the models used for the MAXIMUM CUT and MINIMUM PARTITION problems.

2.2.4.1 Integer Linear Program for MAXIMUM CUT Problem

The following model is taken from [67]. Given a graph $G(V, E)$ such that $V = \{1, 2, 3, \dots, n\}$, $n = |V|$, and A_{ij} being the corresponding Adjacency matrix terms, we have

Objective : $\max \sum_{1 \leq i < j \leq n} x_{ij} A_{ij}$

Constraints :

1. $x_{ij} \leq x_{ik} + x_{kj} \quad (\forall i, j, k \in V)$
2. $x_{ij} + x_{ik} + x_{kj} \leq 2 \quad (\forall i, j, k \in V)$
3. $x_{ij} \in \{0, 1\}$

2.2.4.2 Integer Quadratic Program for MINIMUM PARTITION Problem

Given a set $S = \{w : w \in \mathbb{Z}^+\}$, and $A \subseteq S$, we have

Variables : $x_i = \begin{cases} 1 & \text{if } w_i \in A \\ 0 & \text{if } w_i \notin A \end{cases}$

Objective : $\min \left(\sum_{i=1}^n w_i x_i - \sum_{i=1}^n w_i (1 - x_i) \right)^2$

2.3 Computational Results and Discussions

In this section we first show the performance of the algorithm for the MAXIMUM CUT problem. We compare the results of LogQ with the optimal solution achieved using an integer linear program. We test LogQ on both a quantum simulator and real hardware. Then, the effect of increasing graph density on performance is tested to surpass the sparse examples found in the literature. Finally for MAXIMUM CUT, quantum simulator runs of up to 256 nodes are shown. Then we display the results of the MINIMUM PARTITION problem, which has been solved by converting it to the MAXIMUM CUT problem.

Next, the results from the QUBO method are shown. The MINIMUM PARTITION problem is solved, this time using the QUBO method, and the results are compared with the conversion method.

2.3.1 MAXIMUM CUT

We start by benchmarking the MAXIMUM CUT algorithm against classical methods such as 0-1 integer linear programming and Goemans-Williamson method. All graph instances in this section are generated using the `fast_gnp_random_graph()` function of the `networkx` Python package, with `seed = 0` for all cases.

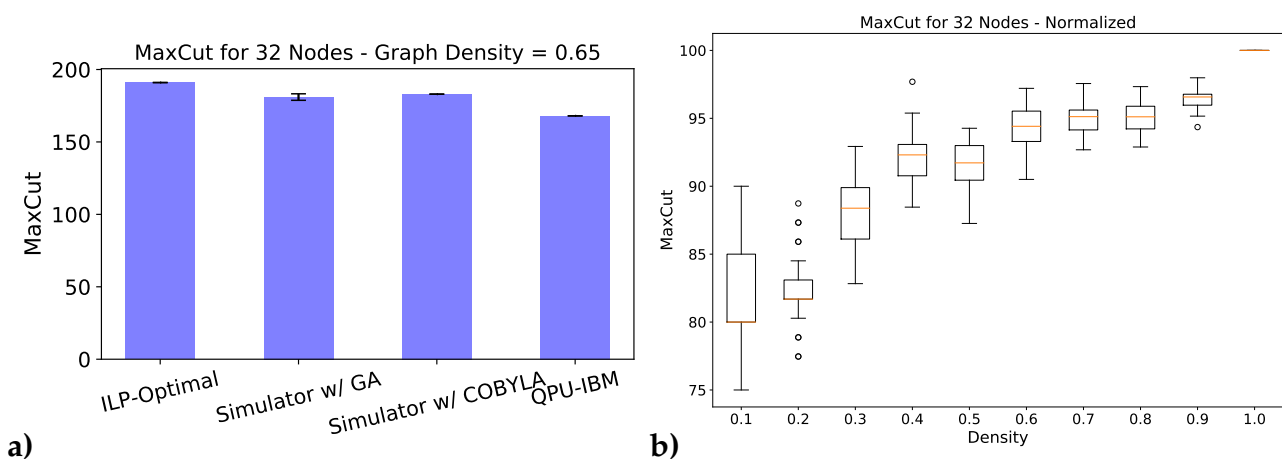


Figure 2.9: **a)** Performance of the algorithm on a 32-Node graph instance. The QPU result is based on a single run while the simulator results are based on 50 runs. **b)** The MAXIMUM CUT of 32-node graphs of varying densities. Optimizer used is Genetic Algorithm (GA). Data is based on 50 runs for each instance and is normalized using the optimal result obtained using ILP.

Figure 2.9a) shows the performance of the algorithm versus the optimal solution obtained using an Integer Linear Program (see section 2.2.4.1). Two different classical optimizers have been used for the runs on the Quantum Simulator. We can see that both classical optimizers give fairly similar results, 90.04% of the optimal for the genetic algorithm and 91.04% of the optimal for COBYLA. While the GA result varies with each run, the results from COBYLA are the same in each run. This can be seen from the fact that the COBYLA plot has a flat error bar. The result from the QPU is slightly worse (83.58% of the optimal), as is expected due to the noise present in the current devices. Note that only a single instance has been considered here as opposed to multiple. This is because running algorithms on real hardware is extremely time consuming due to queue times (wait times).

In Figure 2.9b), 10 randomly generated 32-Node instances are tested with increasing graph density. Here graph density implies the fraction of the total possible edges present

Graph Density	ILP		GW		Quantum Simulator		
	Soln.	Time(s)	Soln.	Time(s)	Soln.	Time(s)	Approx. Ratio (%)
0.30	384	1075	371.4	0.94	343.9	280	89.6
0.35	443	623	428.7	1.00	400.8	272	90.4
0.40	499	997	486.2	1.01	454.3	270	91.0
0.45	556	3600	539.6	1.14	512.8	272	92.2

Table 2.2: 64-Node MAXIMUM CUT benchmarks for Quantum Simulator

Graph Density	QPU		
	Soln.	Time(s)	Approx. Ratio (%)
0.30	282	383	73.4
0.35	365	439	82.4
0.40	380	393	76.1
0.45	446	518	80.2

Table 2.3: 64-Node MAXIMUM CUT benchmarks for QPU

in the graph. For each instance, data was collected for 50 runs, using Genetic Algorithm as the classical optimizer. In addition, a 0-1 integer linear program (ILP) [67] was used to obtain the optimal result of each of the instances. The ILP data is then used to normalize the simulator data. Hence, the data is in the form of percentage of optimal value.

It is seen that the performance improves with an increase in the density of the problem. This might be since in graphs of lower density, the choice including or not including one or two nodes can have a relatively significant impact on the final cost function. For example, the minimum, maximum and average cost function for density 0.1 is (30, 36, 32) and for 0.9 is (234, 243, 239). This shows that the variation of cost function is similar for both densities. This higher instability is why we see a higher dispersion on the results in the lower densities. The important point here is that the performance does not degrade with increasing density, a property which will be useful in the sections to follow.

Graph Density	128 Nodes				256 Nodes			
	GW Mean	GW Upper Bound	Quantum Solution	Ratio with GW Bound (%)	GW Mean	GW Upper Bound	Quantum Solution	Ratio with GW Bound (%)
0.3	1405.6	1567	1305	83.2	5543.5	6159	5066	82.2
0.4	1837.5	2045	1691	82.7	7264.1	8071	6736	83.5
0.5	2249.6	2489	2103	84.5	8120.3	9910	8367	84.4
0.6	2654.1	2981	2546	85.4	10615.1	11794	9967	84.5

Table 2.4: 128 and 256-Node MAXIMUM CUT benchmarks

In order to demonstrate the scalability of the algorithm, we further test the algorithm on problem instances of 64, 128 and 256 nodes (6, 7 and 8 qubits respectively). For the case of 64 node graphs, as shown in Tables 2.2, each instance is run 10 times on the quantum simulator and their mean and standard deviation are shown. The genetic optimizer (GA) is used for

all obtained data. Table 2.5 shows the parameters of GA used in the algorithm runs. The ILP model solved the problem up to optimality (0% gap) for instances with density 0.3, 0.35 and 0.4 while the instance with density 0.45 was solved for 1 hour with a gap of 1.4%. The ILP was run using Gurobi optimizer (version 10.0.1 build v10.0.1rc0) on a computer with 32GB RAM and Apple M2 Pro processor. Also shown are the results of Goemans Williamson (GW) method [32]. Since this is an approximate method, a solution range is shown over 50 runs. We can see that for all cases, the quantum simulator results are nearly or over 90% of the ILP-optimal cut. It is seen again that increase in graph density does not degrade the performance. Table 2.3 shows the QPU results for the same 64 node instances. Figure 2.10 shows the GA convergence plot of one of the quantum simulator runs for an instance of size 64 and density 0.4.

GA Parameter	Value
max_num_iteration	20
population_size	20
mutation_probability	0.1
elit_ratio	0.05
crossover_probability	0.5
parents_portion	0.3
crossover_type	uniform
max_iteration_without_improv	None

Table 2.5: Parameters of the Genetic Algorithm used in the MAXIMUM CUT, MINIMUM PARTITION and MAXIMUM CLIQUE experiments

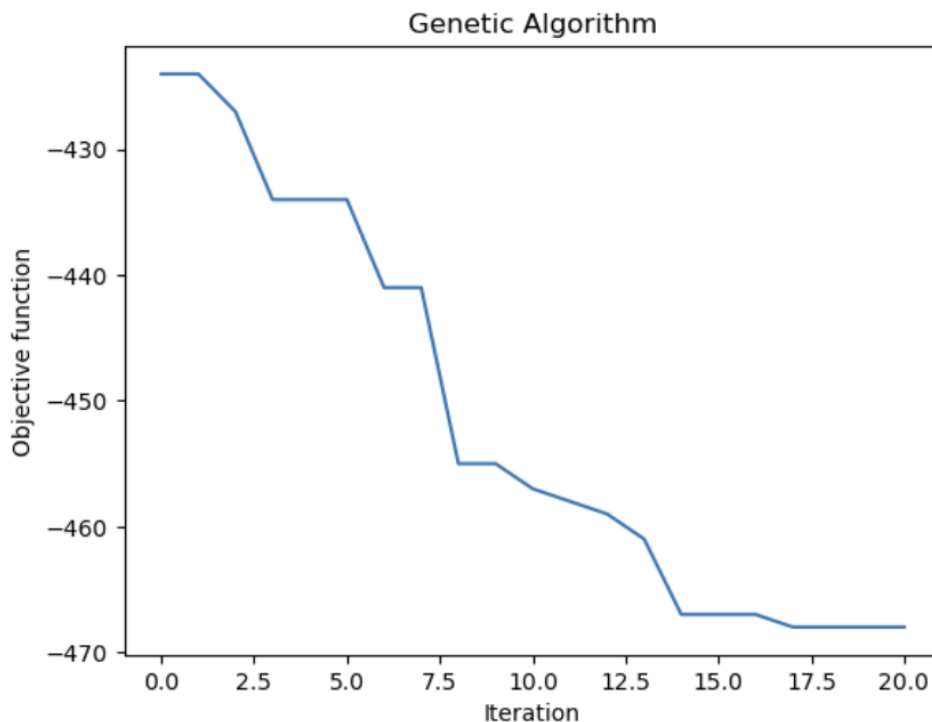


Figure 2.10: Convergence plot of the Genetic Algorithm for a 64-Node MAXIMUM CUT run (on Quantum Simulator).

For 128 and 256 nodes (Table 2.4 and 2.6), only the GW method is used for benchmarking. This is because the ILP took longer than 2 days without converging (Gurobi optimizer) on a PC. The GW mean is based on 50 runs. The quantum solution is compared to the GW upper bound. The GW upper bound is calculated by dividing the worst solution of the GW with 0.878. Table 2.4 shows results using a quantum simulator while Table 2.6 shows results obtained using an IBM quantum computer. The *ibmq_mumbai* backend was used for the instances of size 128 while the *ibmq_guadalupe* was used for the instance of size 256. The results demonstrate the stability of the results (around 84% of the GW upper bound) as the size increases. Figure 2.11 shows the coupling map of *ibmq_mumbai*.

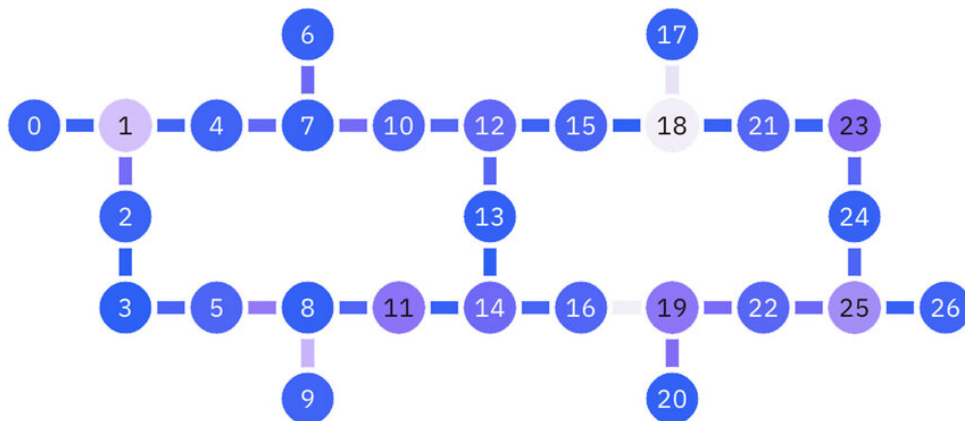


Figure 2.11: Coupling map of *ibmq_mumbai* Quantum Computer.

It is highly time consuming to carry out each run on an IBM QPU due to the significant wait times in addition to the inability to run multiple instances or runs simultaneously. This is the reason we decided to demonstrate only one run per instance where possible. The aim of the QPU runs is not to present a performance analysis of the QPU but merely indicative of the current gap between the quantum simulator and the QPU.

Instance	Q. Sol.	GW Range	% Diff.
Size=128, Density=0.4	1538	1796 - 1864	82.5 - 85.6
Size=128, Density=0.5	2022	2186 - 2271	89.0 - 92.5
Size=256, Density=0.5	8079	8701 - 8880	90.9 - 92.8

Table 2.6: 128 and 256-Node MAXIMUM CUT results using QPU with GA.

It is important to note that the number of GA iterations has a significant impact on the quality of the solution. This is demonstrated in Figure 2.12 with the instances of size 128. On the x-axis, we have the number of GA iterations while on the y-axis we have the MAXIMUM CUT values expressed as a % gap to the mean GW solution (shown in Table 2.4). The solutions improve with an increase in GA iterations. It is expected that for a high enough number of GA iterations, the gap should converge to 0%. However, this can be long and therefore it is often needed to specify a maximum number of GA iterations allowed and accept the best solution we get.

While all the instances tested here are a power of 2 to maximize the number of vertices for a given number of qubits, the method works for any intermediate size. To demonstrate this, Table 2.7 shows MAXIMUM CUT results for instances of size 50 on a quantum simulator. The ILP solutions are optimal.

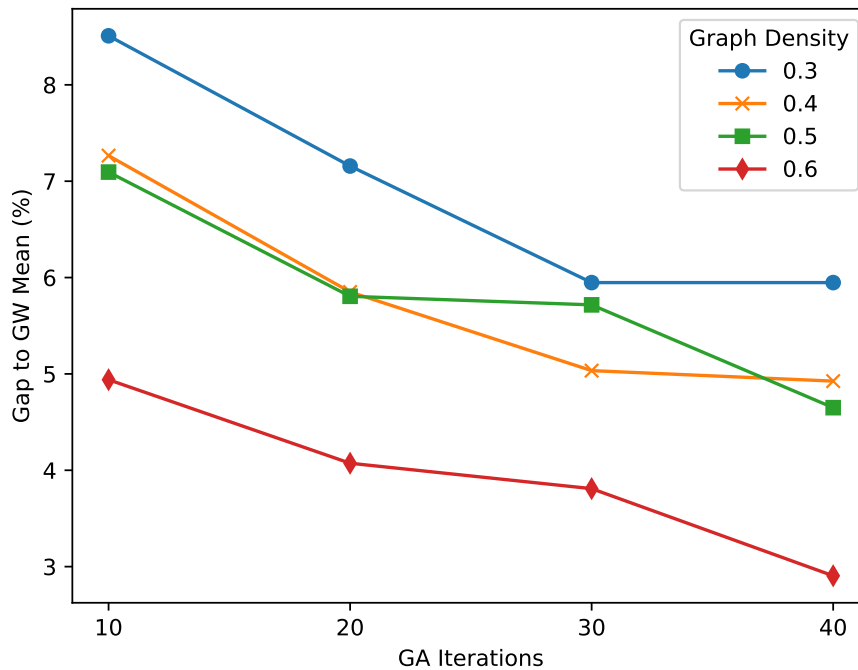


Figure 2.12: Evolution of MAXIMUM CUT solutions with increasing GA iterations on 128-Node instances.

Graph Density	ILP Solution	Quantum Simulator	
		Soln.	Approx. Ratio (%)
0.30	238	217.5	91.4
0.40	302	284	94.0

Table 2.7: 50-Node MAXIMUM CUT results for Quantum Simulator. Data is an average over 10 runs.

2.3.2 MINIMUM PARTITION as a conversion from MAXIMUM CUT

As described in Section [2.2.2.1](#), the number partitioning problem can be directly converted into the MAXIMUM CUT problem. The graphs hence formed are weighted fully dense graphs.

For the instances, all the numbers used were random integers between 1 and 100. Tests were carried out on the quantum simulator as well as on real hardware from IBM. The classical optimizer used is GA. Details of the GA parameters are given in Table [2.5](#).

The results of partition differences have been normalized in the following manner. For a problem with N numbers, if the partition difference is p , then the normalised difference is $p_{norm} = \frac{50N - p}{50N}$. All our numbers are random integers between 1 and 100, hence 50.5 on an average. For simplicity we use 50 in p_{norm} .

The optimal value for each instance is obtained using the Integer Quadratic model described in section [2.2.4.2](#).

Figure [2.13](#) displays the performance of 32, 64, and 128-number MINIMUM PARTITION converted to MAXIMUM CUT. For instances of all sizes, we have a near-optimal mean value

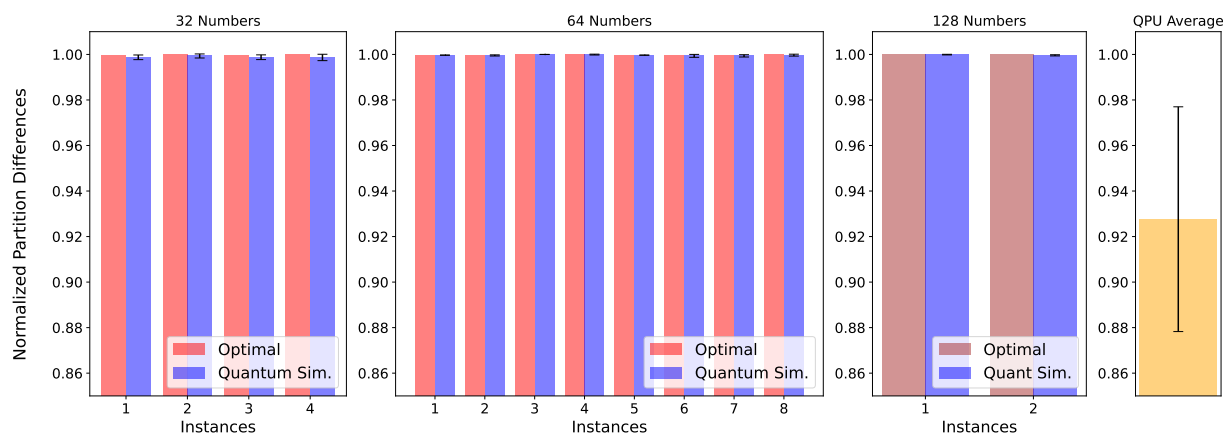


Figure 2.13: The difference between partition sets for 32, 64 and 128 Numbers. Each instance was run on the quantum simulator with GA, 100 times for 32 numbers, 10 times for 64 numbers and 4 times for 128 numbers. The QPU data is the aggregate of single runs for every 32 and 64 Number instance.

and a very small dispersion using the quantum simulator. Moreover, for problem sizes 32 and 64, despite the fact that the MINIMUM PARTITION problem leads to a complete graph MAXIMUM CUT problem, actual QPUs are able to demonstrate an approximate solution of good quality. This further shows that the performance of the algorithm is not adversely affected by dense graphs.

All QPU runs in this section are done on *ibmq_mumbai*.

2.3.3 MAXIMUM CLIQUE as a conversion from MAXIMUM CUT

The MAXIMUM CLIQUE problem can be converted to the MAXIMUM CUT problem by first converting it to the MAXIMUM 2-SAT (2.2.2.2) and then from the MAXIMUM 2-SAT to MAXIMUM CUT (2.2.2.3).

After the conversion, a n -node MAXIMUM CLIQUE problem requires the solution of a $2(n+1)$ -node MAXIMUM CUT. Table 2.8 shows results for various instances run on a quantum simulator. The classical optimizer used is GA, whose Details of the GA parameters are given in Table 2.5. The optimal solutions for all the instances are calculated using the *numpy* package in python. It is seen that in half of the instances, the best solution is the optimal solution. Note that we do not always find a feasible solution. In other words, the solution found is not always a clique. In those cases, initially, a dense subgraph is output by the algorithm and then we use post processing to remove the nodes with lowest degree iteratively until the solution is a clique.

2.3.4 MAXIMUM WEIGHTED INDEPENDENT SET using QUBO method

In this section, results of the MAXIMUM WEIGHTED INDEPENDENT SET problem solved using the QUBO method (section 2.2.3.1) is presented.

For each figure the performance of the algorithm is shown. The data is normalized using the optimal solution found using the commercial CPLEX solver (version 22.1.0).

Figure 2.14 shows the data for graphs of size 32, 64 and 128. The data for 32, 64 and 128 nodes is based on 50, 50 and 10 runs respectively. The data represented only takes into account the feasible solutions produced. For graphs of size 32, the mean values for all instances are above 80% and the best obtained result is optimal for every instance. For

Instance (Size, Density)	No. of Qubits	Best Solution	Worst Solution	Average Solution	Optimal Solution
31, 0.3	6	4	3	3.38	4
31, 0.4	6	5	3	3.92	5
31, 0.5	6	6	3	4.46	6
31, 0.6	6	7	4	5.34	8
31, 0.7	6	8	5	6.26	9
63, 0.5	7	8	6	6.5	8
63, 0.6	7	8	6	7.2	10
63, 0.7	7	11	8	9.3	12

Table 2.8: MAXIMUM CLIQUE results using quantum simulator with GA. The data is based on 50 runs and 10 runs for instances of size 31 and 63 respectively.

graphs of size 64, the mean values for all instances are above 60% and the best obtained result is on an average over 80%. For 128-node graphs, the solutions are slightly degraded in comparison. Note, however, that the data for 128-node instances is based only on a few runs. Moreover, the performance also depends on the number of GA iterations used in the algorithm run. The details of the GA parameters used are given in Table 2.9.

Genetic Algorithm Parameter	Instance Size		
	32	64	128
max_num_iteration	50	100	200
population_size	20		
mutation_probability	0.1		
elit_ratio	0.05		
crossover_probability	0.5		
parents_portion	0.3		
crossover_type	uniform		
max_iteration_without_improv	None		

Table 2.9: Parameters of the Genetic Algorithm used in the MAXIMUM WEIGHTED INDEPENDENT SET experiments

Table 2.10 shows how the performance varies depending upon the number of GA iterations used. For this table, the MAXIMUM WEIGHTED INDEPENDENT SET Instance 4 of size 64 has been taken. An increase in the number of GA iterations not only improves the performance but also the percentage of feasible results.

2.3.5 A comparative study of time taken by the simulator and the QPU

In Table 2.11 and Figure 2.15, the time taken to run the algorithm for different MAXIMUM CUT instance sizes is compared. While the quantum computer still takes a significant amount of time to solve the problem, the time taken does not increase exponentially as in the case of the simulator. As we move towards larger instances, we reach a point where it is quicker to run a problem on a QPU than using a simulator.

Note that the QPU time here does not take into account the queue time or waiting time for the QPU runs. The real-world time was several hours or even several days for the largest

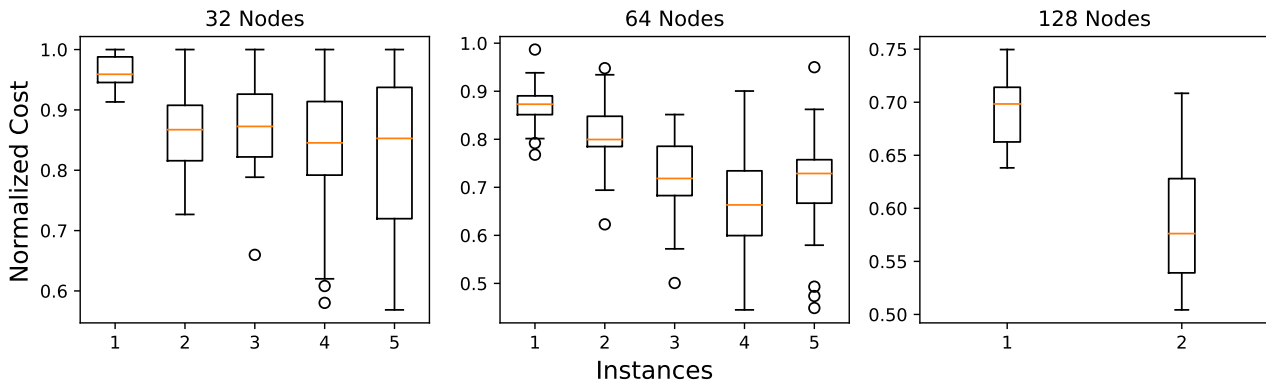


Figure 2.14: MAXIMUM WEIGHTED INDEPENDENT SET problem for 32, 64 and 128-node graphs using a quantum simulator. Each instance was run on a quantum simulator with GA 50 times for graph sizes of 32 and 64 and 10 times for graph size of 128.

Size	GA iterations	Solution as % of Optimum	% of Feasible Solutions
64	50	54.8	60
64	100	66.6	96
64	200	77.8	100

Table 2.10: MAXIMUM WEIGHTED INDEPENDENT SET results demonstrating the relationship between GA iterations and performance. The Instance 4 of size 64 is used for this table. The performances are an average over 10 runs.

run instance. This prevented us from running larger instances on the quantum computer.

N	QPU(minutes)	Quantum Simulator(minutes)
32	1.7	3
64	9	52
128	45	222
256	112	3202

Table 2.11: Data for time taken for various instance sizes in the the QPU and in the quantum simulator

2.4 Conclusion

In this chapter, we investigated and further developed methods to logarithmically encode combinatorial optimization problems on a quantum computer. We begin by explaining the LogQ encoding for the MAXIMUM CUT problem in detail. To help readers better understand the intricacies and novelty of the algorithm, we then demonstrate step-by-step all the calculations related to a run of the algorithm on a small example. In order to analyze the algorithm, we perform several runs of the LogQ algorithm for the MAXIMUM CUT problem

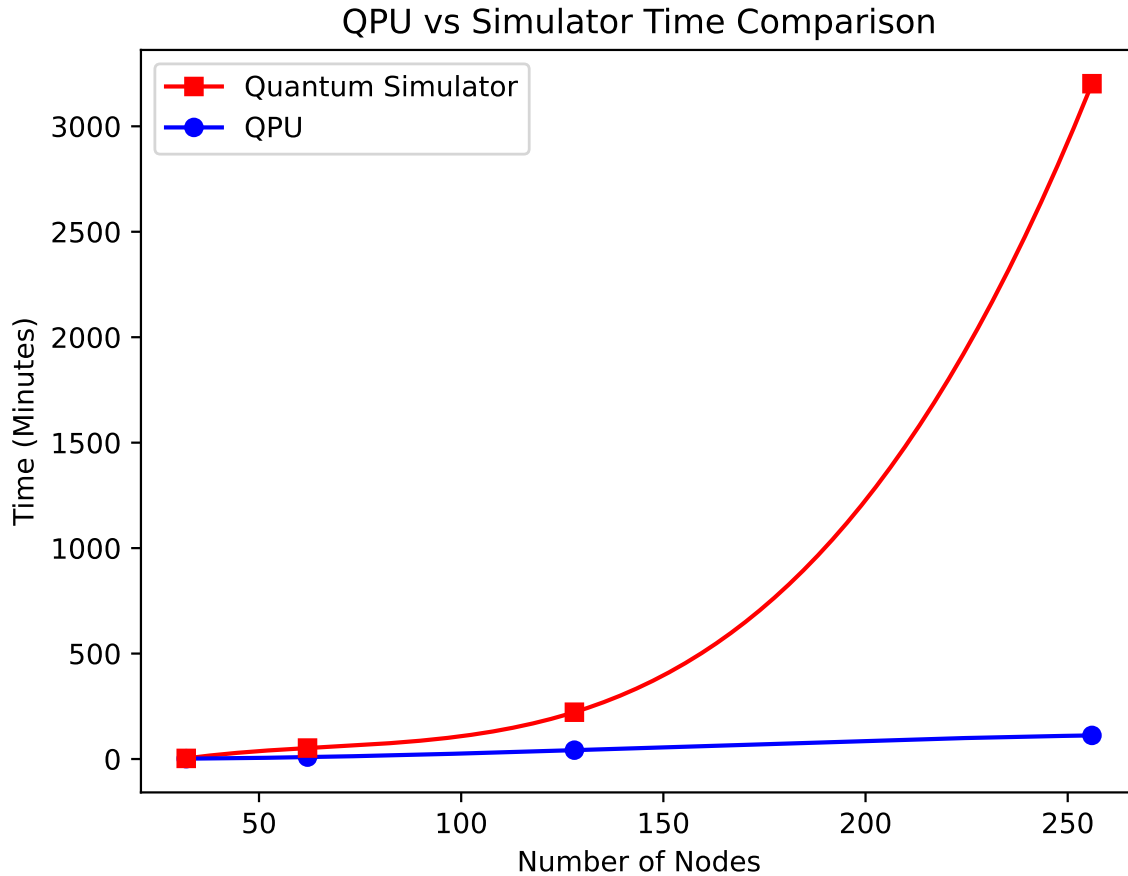


Figure 2.15: Plot demonstrating the time taken by the quantum simulator versus the time taken to solve the same instance on real hardware

with various instances, on the quantum simulator as well as real hardware, using different classical optimizers like COBYLA and the genetic algorithm.

We then reformulate a number of NP-hard combinatorial optimization problems into the MAXIMUM CUT problem, either directly or indirectly and solve it on a real quantum computer. We take the MINIMUM PARTITION problem as an example and solve it by using a reduction as mentioned in section [2.2.2.1](#). This is possible since the algorithm is largely unaffected by increasing the density of the MAXIMUM CUT graph in question, since the MINIMUM PARTITION problem converts into a weighted fully-dense graph. Some performance benchmarks of the partition problem have been presented.

We then proceed to present a more general formulation inspired from the structure of the MAXIMUM CUT algorithm. We see that instead of using the Laplacian, we can use the QUBO matrix of a problem in order to solve it. We introduce the sQUBO representation of the QUBO matrix for it to be compatible with the algorithm. This therefore opens up the applicability of the algorithm to a wide range of algorithms. The MAXIMUM WEIGHTED INDEPENDENT SET problem is solved using its sQUBO matrix.

To our knowledge, it is the first time that graph problems of such sizes (256 MAXIMUM CUT, 64 MINIMUM PARTITION) have been executed on real universal gate-based quantum computers.

A Hybrid Quantum-assisted Column Generation Algorithm for Fleet Conversion

3.1 Introduction

In the previous chapter, the LogQ encoding to solve various combinatorial optimization problems was introduced. We showed that using this method we can encode instances of much larger sizes with relatively small quantum computers. In this chapter, we aim to apply this method to an industrial use case.

Fleet conversion is the process of transitioning a fleet of vehicles to more sustainable and environmentally friendly alternatives. With the growing recognition of the detrimental effects of traditional fossil fuel powered vehicles on the environment and the need to mitigate climate change, businesses and organizations are increasingly looking for ways to reduce their carbon footprint and operate more efficiently. The transportation sector is one of the largest contributors to greenhouse gas emissions, primarily due to their reliance on fossil fuels. By transitioning fleets to electric or hybrid vehicles, large companies such as TotalEnergies can significantly reduce their carbon emissions. Beyond the environmental benefits, fleet conversion also offers compelling cost-saving opportunities for businesses.

In the fleet conversion problem, a certain number of tours need to be carried out between several locations. In order to carry out these tours we have at our disposal several vehicles of different models. Each vehicle model has an associated cost. On top of the capital expenditure corresponding to the purchase of one vehicle of one model, this cost may also capture the environmental cost – e.g. the carbon footprint; the cost of operation – e.g. energy usage, or both. The objective is to minimize the total cost of carrying out all the tours including capital and operational expenditures. Therefore, fleet conversion goes beyond simply choosing the best possible vehicles and also incorporates sharing the same vehicles for multiple tours when possible, thereby reducing the cost.

In chapter 2, the LogQ encoding to treat Quadratic Unconstrained Binary Optimization (QUBO) [31], [68]–[70] problems using *logarithmically* fewer qubits has been demonstrated. In this chapter, we use column generation [71]–[75] to describe our problem as a coordinator problem and several sub-problems henceforth referred to as workers. The worker problem in our case is the Maximum Weighted Independent Set (MWIS) problem which can be represented as a QUBO problem. We propose an algorithm that handles the coordinator problem

using a commercial linear program solver Gurobi and the worker problems using a quantum solver based on the LogQ encoding. In our experiments, we solve instances up to a size of 64 tours using only 7 qubits to represent the MWIS workers. This shows that the method is compatible with the quantum computers of the NISQ era.

The chapter is structured as follows. In section 3.2.1 the fleet conversion problem is defined. In section 3.2.2 the problem is stated in the form of a graph problem followed by section 3.2.3 where the column generation algorithm is described. In section 3.2.4 and 3.2.5, we describe the quantum model to solve the sub-problems and how we can use the quantum solver and classical solver together to develop a quantum-assisted algorithm. In section 3.3 we present the experimental results. In section 3.4 we present an extension to the algorithm and finally in section 3.5 we have the discussion and conclusion.

The work presented in this chapter has been submitted in a journal and the preprint is available in arxiv [76]. It has been also been presented in ROADEF 2024 [77].

3.2 Problem Statement and Methods

In this section, we present the Fleet Conversion Problem as well as its formulation as a weighted graph coloring problem. We then reformulate the problem using the definition of independent sets and build a column generation approach to solve it. The column generation approach uses sub-problems that compute max-weighted independent sets, further brought together iteratively to build a global graph coloring solution. We then demonstrate how a quantum algorithm can be crafted to solve these sub-problems and integrate the column generation procedure.

3.2.1 Statement

Let L be a set of locations, K a set of tours, V a set of available vehicle models, and C a set of physical vehicles, henceforth referred to as *color*.

Notations

For any physical vehicle c of a certain model v , let us write $v(c) := v$. Tours are carried out from one location to another. Let w_v^k be the cost to assign a vehicle model v to any tour k . Assigning a model v to a tour k means that the tour k has to be carried out by a physical vehicle (color) from model v . To do so, some color c such that $v(c) = v$ has to be assigned to tour k . The cost w_v^k captures the operational expenditures incurred by performing tour k with a vehicle model v . Since every color belongs a specific vehicle model, let the cost to assign the physical vehicle (color) c to any tour k be $\Gamma_c^k := w_v^k$ when $v = v(c)$. We also define a cost $\gamma_{v(c)} > 0$ for using a color c at least once. Clearly, γ_v represents the cost of purchase of one physical vehicle of model v and we reasonably assume this cost is independent of the color (all vehicles of the same model are equivalent). Thus, without risk of confusion, we define $\gamma_c := \gamma_{v(c)}$.

Each tour $k \in K$ is described by the tuple $(t_d^k, t_a^k, l_d^k, l_a^k, A^k)$, corresponding respectively to the departure time, arrival time, departure location and arrival location, and a set of authorized vehicle models of the tour. The time to travel from location i to location j (TT_{ij}) can be computed using the distance matrix of the locations. This matrix is used to derive the time needed to relocate any physical vehicle from the arrival location of a tour to the departure location of another tour in case these tours are meant to be assigned to the same color.

Incompatibilities

In the Fleet Conversion Problem, we have two types of incompatibilities:

- *tour-color incompatibilities*: a tour k cannot be assigned to a color from a certain model v . In practice, this incompatibility can model constraints in urban mobility such as the forbidden penetration of internal combustion engines into low emission zones or simply drivers' personal preferences (manual *vs* automatic, plug-in hybrid *vs* electric, *etc*). Let A^k be the set of authorized colors for tour k . Specifically, a color c can be assigned to tour $k \in K$ only if $v(c)$ is in the set of allowed models A^k for that tour.
- *tour-tour incompatibilities*: two different tours cannot share the same color (physical vehicle) because they occur at the same time, or because their departure and/or arrival locations make it impossible to transition in an acceptable time without perturbing the global schedule. Let $\mathcal{I} \subset \binom{K}{2}$ be the set of unordered couples with a tour-tour incompatibility¹.

First formulation

The aim of the Fleet Conversion Problem is to minimize the overall cost $\sum_c \gamma_c y_c + \sum_{k,c} \Gamma_c^k x_c^k$, where

$$x_c^k = \begin{cases} 1 & \text{if color } c \text{ is assigned to tour } k \\ 0 & \text{otherwise,} \end{cases} \quad (3.1)$$

and

$$y_c = \begin{cases} 1 & \text{if color } c \text{ is purchased} \\ 0 & \text{otherwise.} \end{cases} \quad (3.2)$$

This can be achieved by preferring colors c with low values of γ_c and also by assigning multiple compatible tours to a single color while choosing the minimal values of Γ_c^k if possible. This is to be done in such a way that all tours within K are assigned to a compatible color.

Therefore, the Fleet Conversion Problem can be expressed as the following optimization problem:

$$\min \sum_{c \in C} \gamma_c y_c + \sum_{k \in K, c \in C} \Gamma_c^k x_c^k \quad (3.3)$$

$$\text{s.t. } \sum_c x_c^k \geq 1 \quad (\forall k \in K) \quad (3.4)$$

$$x_c^k \leq y_c \quad (\forall c \in C)(\forall k \in K) \quad (3.5)$$

$$x_c^k + x_c^{k'} \leq 1 \quad (\forall \{k, k'\} \in \mathcal{I})(\forall c \in C) \quad (3.6)$$

$$x_c^k = 0 \quad (\forall k \in K)(\forall c \in C)(v(c) \notin A^k) \quad (3.7)$$

$$y_c, x_c^k \in \{0, 1\}. \quad (3.8)$$

Equation (3.4) states that a tour k has to be assigned to at least one² color, whereas Equation (3.5) forbids the assignment of tours to a color unless the color is purchased. Equa-

¹For any set X and any integer n , $\binom{X}{n} := \{I \subset X \mid |I| = n\}$.

²In fact, *exactly one* would be more appropriate than *at least one*. However, as the variables x_c^k are penalized by a cost in the objective, the two formulations are actually equivalent and at optimum, this constraint is actually active

tion (3.6) forbids incompatible tours to share the same color and Equation (3.7) avoids tour-color incompatibilities.

Problem (3.3)–(3.8) is thus a Mixed Integer Linear Program (MILP) that can be solved with for instance a commercial solver. However, as it is notoriously NP-complete as from $|C| \geq 3$, solver time performance will worsen quickly with problem size. For this reason, as a proof-of-concept, we reformulate Problem (3.3)–(3.8) and design an algorithm that will scale better.

3.2.2 Formal description as a Graph Problem

Let $G = (K, E)$ be a graph where the nodes of the graph are the tours and the edges $(k, k') \in E$ of the graph denote the *incompatibility* of the tours k and k' . Note that the nodes of the graph DO NOT represent a specific location but aggregate a complete tour (including starting location, travel time, destination location). With the notations of the last paragraph, this means that we let $E := \mathcal{I}$. Two tours i and j are compatible if their time-windows do not overlap and we have enough time to travel from the arrival location of tour i to the departure location of tour j , without loss of generality if tour i occurs before tour j . Formally, we define $E = \{(i, j) | \forall i, j \in K, t_a^i + TT_{ij} > t_d^j\}$. For every tour k , we have a list of allowed models A^k .

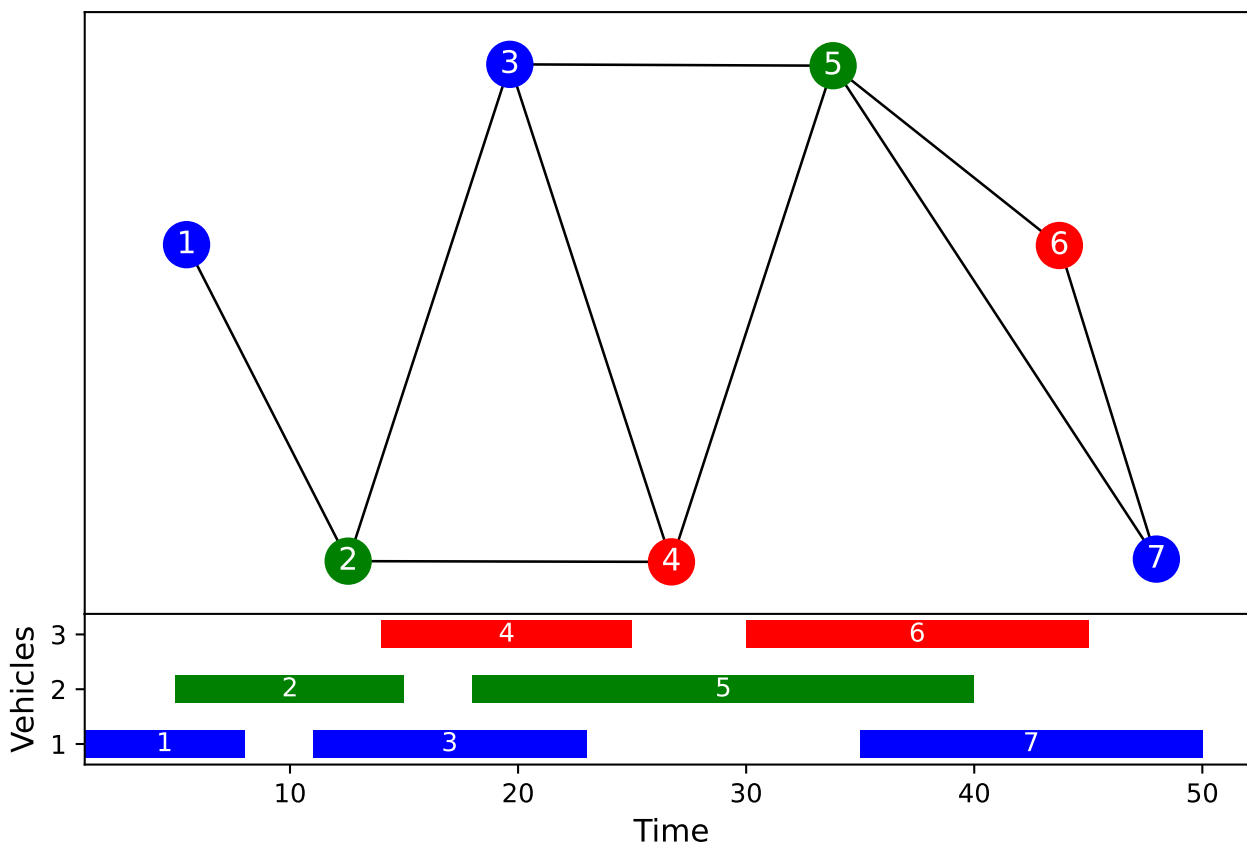


Figure 3.1: Incompatibility Graph generated from the time windows of tours. The lines represent the time windows of 7 tours. Colors *blue*, *red* and *green* are used to assign vehicles to tours taking into account their incompatibility.

Figure 3.1 illustrates how an incompatibility graph can be constructed using time windows of several tours, along with a possible coloring of the graph. For example, since the

tour 3 overlaps with tours 2, 4 and 5, the node 3 has edges with node 2, 4 and 5. For a detailed description of all the variables used in this section and the following section, see Table [3.1](#).

Definition 3.1 (Independent set) *An independent set in $G = (K, E)$ is a subset $I \subseteq K$, such that for any two vertices $k, k' \in I$, $(k, k') \notin E$. In other words, an independent set of our graph is a subset of tours that are all compatible with each other to share colors.*

With the definition of an independent set, the following fact is straightforward.

Fact 3.1 *Let (x, y) be any solution of Problem [\(3.3\)](#)–[\(3.8\)](#). Then, for all $c \in C$ such that $y_c = 1$, $\{k | x_c^k = 1\}$ forms an independent set of G .*

An independent set I can be derived into an allocation (a_I, c_I) , where $c_I \in C$ is a color and a_I is the indicator vector of set I : $a_I^k = 1$, if $k \in I$, and 0 otherwise. For any independent set I and color c , let $\Gamma_{I,c} := \gamma_c + \sum_{k \in I} \Gamma_c^k$ denote the cost of allocation (a_I, c) . Here, we consider only *feasible* allocations, specifically respecting tour-color compatibility. We can therefore define the set of feasible allocations Λ formally as :

$$\Lambda = \{(a_I, c_I) | I \text{ independent set of } G \text{ s.t. } \forall k \in K, a_I^k = 1 \Rightarrow v(c_I) \in A^k\}. \quad (3.9)$$

For lighter notations, we refer to the element $(a_I, c_I) \in \Lambda$ only as $I \in \Lambda$, keeping in mind that the independent set I , considered as an allocation, comes with a color c_I . In particular, we keep in mind that if $c \neq c'$, and even if $v(c) = v(c')$, (a_I, c) and (a_I, c') are two different allocations. Formally, let $\Gamma_I := \Gamma_{I, c_I}$.

The problem can then be equivalently formulated as:

$$\min \sum_{I \in \Lambda} \Gamma_I x_I \quad (3.10)$$

$$\text{s.t. } \sum_{I \in \Lambda} a_I^k x_I \geq 1 \quad (\forall k \in K) \quad (3.11)$$

$$x_I = \begin{cases} 1, & \text{if independent set } I \text{ chosen} \\ 0, & \text{otherwise} \end{cases} \quad (3.12)$$

Note that constraint [\(3.11\)](#) is equivalent to the constraint [\(3.4\)](#). Also, beware that the variables defined in equation [\(3.12\)](#) and equation [\(3.1\)](#) are not the same.

With the above reformulation, the problem looks considerably smaller. We have indeed only one constraint per tour. However, we need to generate the set Λ in order to solve this problem. The set Λ contains all the feasible independent sets of the graph, that is, independent set of the graph coupled with colors that are compatible with all nodes therein. Remember that if I is a non-empty independent set of G and $c \neq c'$ are two different colors, then (I, c) and (I, c') are two distinct elements of Λ as they describe two different conversion solutions. We thus have a very large number of variables x_I . Nevertheless, it is clear that only a small number of those variables should be non-zero at optimum. Indeed, at worst (in terms of number of independent sets activated), no sharing of color is possible and each tour has a dedicated vehicle, which corresponds to $|K|$ variables x_I activated. This is why an interesting approach here is column generation, where independent sets are generated dynamically while the solution converges to an optimum.

3.2.3 The Column Generation Algorithm

In this paragraph, we design the column generation procedure to solve our problem. First, we derive an extension of the problem that makes it always feasible. To do so, we permit the algorithm to reject tours from the solution. Second, we relax all integrity constraints and actually solve the LP-relaxation of our problem. Once an LP-optimal solution is found by the column generation, one can summon any type of rounding algorithm to build (if needed) an integer-feasible solution from the relaxed solution. For instance, see [78]. We focus this work on the problem of finding the optimal LP-feasible solution for our problem.

Problem extension

For all $k \in K$, let r_k be a binary variable that states whether tour k is rejected from the solution or not:

$$r_k = \begin{cases} 1 & \text{if } k \text{ is rejected} \\ 0 & \text{otherwise.} \end{cases} \quad (3.13)$$

Let R be a sufficiently large real number. We now consider the new optimization problem:

$$\min \sum_{I \in \Lambda} \Gamma_I x_I + R \sum_{k \in K} r_k \quad (\text{CP})$$

$$\text{s.t. } \sum_{I \in \Lambda} a_I^k x_I + r_k \geq 1 \quad (\forall k \in K) \quad (\text{CP-1})$$

$$x_I, r_k \geq 0 \quad (\forall I \in \Lambda)(\forall k \in K) \quad (\text{CP-2})$$

When the binary constraints are enforced on x_I and r_k , and if R is sufficiently large, it is clear that Problems (CP) and (3.10)–(3.11) are equivalent. The dual program [79] of (CP) reads :

$$\max \sum_{k \in K} \mu_k \quad (\text{D})$$

$$\text{s.t. } \sum_{k \in K} a_I^k \mu_k \leq \Gamma_I \quad (\forall I \in \Lambda) \quad (\text{D-1})$$

$$\mu_k \geq 0 \quad (\forall k \in K) \quad (\text{D-2})$$

Restriction and generation

Let $\Lambda' \subset \Lambda$ be an arbitrary subset of allocations. One can form the primal dual pair of problems:

$$\min \sum_{I \in \Lambda'} \Gamma_I x_I + R \sum_{k \in K} r_k \quad (\text{RCP})$$

$$\text{s.t. } \sum_{I \in \Lambda'} a_I^k x_I + r_k \geq 1 \quad (\forall k \in K) \quad (\text{RCP-1})$$

$$x_I, r_k \geq 0 \quad (\forall I \in \Lambda')(\forall k \in K) \quad (\text{RCP-2})$$

$$\max \sum_{k \in K} \mu_k \quad (\text{RD})$$

$$\text{s.t. } \sum_{k \in K} a_I^k \mu_k \leq \Gamma_I \quad (\forall I \in \Lambda') \quad (\text{RD-1})$$

$$\mu_k \geq 0 \quad (\forall k \in K) \quad (\text{RD-2})$$

Let F (resp. F') denote the feasible set of (CP) (resp. (RCP)) and G (resp. G') be the feasible set of (D) (resp. (RD)). It is clear that $F' \subset F$ and $G \subset G'$. Furthermore, for any $\Lambda' \subset \Lambda$, (RCP) is linear, feasible and bounded. Therefore, strong-duality applies [79][80]. This means that (RCP) and (RD) are both feasible and a primal-dual pair of solutions (x', μ') exists, where x' solves (RCP), μ' solves (RD). Furthermore, it means we have the equality:

$$\sum_{I \in \Lambda'} \Gamma_I x'_I + R \sum_{k \in K} r'_k = \sum_{k \in K} \mu'_k. \quad (3.14)$$

The column generation is based on the following fact:

Fact 3.2 Let $\Lambda' \subset \Lambda$. Let (x^*, μ^*) and (x', μ') be primal-dual optimal couples for (CP)–(D) and (RCP)–(RD) respectively. By definition:

- $x^* \in F$
- $x' \in F' \subset F$
- $\mu^* \in G \subset G'$
- $\mu' \in G'$

Suppose that $\mu' \in G$.

Then, $x' \in F$ and x' is optimal for (CP).

Indeed, if $\mu' \in G$, by definition of μ^* , we know that $\sum_{k \in K} \mu'_k \leq \sum_{k \in K} \mu_k^*$. On the other hand, as $G \subset G'$, by definition of μ' , we have $\sum_{k \in K} \mu'_k \geq \sum_{k \in K} \mu_k^*$. Thus, we have equality. By strong duality, this means that

$$\sum_{I \in \Lambda'} \Gamma_I x'_I + R \sum_{k \in K} r'_k = \sum_{k \in K} \mu'_k \quad (3.15)$$

$$= \sum_{k \in K} \mu_k^* \quad (3.16)$$

$$= \sum_{I \in \Lambda'} \Gamma_I x_I^* + R \sum_{k \in K} r_k^* \quad (3.17)$$

Therefore, if μ' is feasible in (D), then x' is an optimal solution of (CP).

For μ' to be feasible in (D), the following constraint must hold:

$$\sum_k a_I^k \mu'_k \leq \Gamma_I \quad \forall I \in \Lambda \quad (3.18)$$

Therefore, the existence of violated constraints (3.18) means that the current x' is not the optimum and that new columns (allocations) can be added to improve the solution. We can therefore try to find an independent set that minimizes the *reduced cost* $\Gamma_I - \sum_k y_I^k \mu'_k$. If this reduced cost is negative then the independent set I used to obtain this negative reduced cost violates (3.18) and can therefore be added to the set of independent sets in RCP Λ' .

For an allocation I , we have:

$$\Gamma_I = \sum_k a_I^k \Gamma_c^k \quad (3.19)$$

Therefore the reduced cost to minimize is:

$$\begin{aligned} \sigma &= \Gamma_I - \sum_k a_I^k \mu'_k \\ &= \sum_k a_I^k \Gamma_c^k - \sum_k a_I^k \mu'_k \\ &= \sum_k a_I^k (\Gamma_c^k - \mu'_k) \end{aligned} \quad (3.20)$$

In order to convert this into a maximization problem, we can simply change the sign. The problem therefore reads:

$$\max \sigma = \sum_k a_I^k (\mu'_k - \Gamma_c^k) \quad (3.21)$$

By definition, $(a_I^k)_{k \in K}$ is a vector denoting an independent set, and $\mu'_k - \Gamma_c^k$ can be seen as numerical weights for every color c . This is therefore a Maximum Weighted Independent Set problem. For the rest of the chapter, this problem will be our worker problem (WP). The MWIS problem can be defined as follows.

$$\begin{aligned} \max \sigma &= \sum_k y_k (\mu'_k - \Gamma_c^k) && \text{(WP)} \\ \text{s.t. } y_k + y_j &\leq 1 && (\forall (k, j) \in E) \quad \text{(WP-1)} \\ y_k &\in \{0, 1\} && (\forall k \in K) \quad \text{(WP-2)} \end{aligned}$$

A solution of Problem (WP) defines an allocation (a_I, c) where $I := \{k \in K | y_k = 1\}$ and $a_I^k = y_k$. Note that there is one worker problem per color, and, as in our problem, colors from the same model are equivalent, there is actually one problem per vehicle model. Therefore, each allocation produced by a worker represents a single physical vehicle (color) along with all the tours assigned to it. Given the above problem definitions, we can define the algorithm to solve RCP as described in Algorithm 4

3.2.4 Quantum model for the MWIS problem

The MWIS problem can be represented in the form of a QUBO. Our problem in question is the problem WP. For simplicity let $\mu'_k - \Gamma_c^k = w_k$. The objective function is therefore $\sum_k y_k w_k$. The constraint (WP-1) can be incorporated into the objective function as a penalty [31] as follows. The real number P is the penalty strength. Note that the variable y in this section is not related to the other variables y that have appeared previously.

$$\mathcal{C} = \sum_k y_k w_k - P \left(\sum_{(i,j) \in E} y_i y_j \right) \quad (3.22)$$

The transformation of the constraint (WP-1) into the penalty term can be understood using the truth table shown in Table 3.2. When the constraint is violated (*False*), the value of $y_i y_j = 1$ and hence the penalty term is added.

Variables	Description
k	Denotes a tour
K	Set of all tours
A^k	Set of allowed vehicles for tour k
r_k	Binary variable that states whether tour k is rejected from the solution or not
R	Large real number
x_I	Binary variable to decide whether the independent set I is in the solution
Γ_I	Cost of choosing the independent set I
a_I^k	vector denoting the independent set I . It is 1 if node $k \in I$ and 0 otherwise
Λ	Set of all feasible independent sets
μ_K	Dual variable of x_I
Λ'	Subset of feasible independent sets
x^*	Optimal solution of CP
μ^*	Optimal solution of D
x'	Optimal solution of RCP
μ'	Optimal solution of RD

Table 3.1: Description of variables and symbols

Algorithm 4: Column Generation algorithm for RCP

Input: Incompatibility graph $G(K, E)$

- 1 Define R such that $R > \max \Gamma_I + 1$
- 2 $\Lambda' \leftarrow \emptyset$
- 3 **while** *True* **do**
- 4 Solve RCP and get primal dual couple (x', μ')
- 5 **for every model** v **do**
- 6 Solve WP
- 7 **if** $\sigma > 0$ **then**
- 8 Let the optimal solution be $I = (y_I, c_I)$.
- 9 $\Lambda' \leftarrow \Lambda' \cup \{I\}$.
- 10 **end**
- 11 **end**
- 12
- 13 **if** Λ' *has been modified* **then**
- 14 continue
- 15 **end**
- 16 **else**
- 17 Current solution x' is the optimal solution
- 18 break
- 19 **end**
- 20 **end**
- 21
- 22 Apply rounding algorithm to transform the relaxed solution x' into a binary one.

Since, $y \in \{0, 1\}$, we can set $y_k^2 = y_k$. Therefore:

y_i	y_j	$y_i + y_j$	$y_i + y_j \leq 1$	$y_i y_j$
0	0	0	True	0
0	1	1	True	0
1	0	1	True	0
1	1	2	False	1

Table 3.2: Truth Table to demonstrate the equivalence of constraint (WP-1) and the penalty term used.

$$C = \sum_k y_k^2 w_k - P \left(\sum_{(i,j) \in E} y_i y_j \right) \quad (3.23)$$

Since C is in the form of a QUBO problem, we can represent it in the following form:

$$C = y^T Q y \quad (3.24)$$

where y is the independent set vector and Q is the QUBO matrix.

As described in chapter 2, a QUBO problem can be represented on a quantum computer using only a logarithmic number of qubits using the LogQ encoding. We shall use this algorithm to solve WP. Following are some of the main aspects of the algorithm. Our main aim here is to represent (3.24) on a quantum computer.

1. A N -qubit parameterized state $|\Psi(\theta)\rangle$ is created as follows:

$$|\Psi(\theta)\rangle = U(\theta) H^{\otimes N} |0\rangle^{\otimes N} \quad (3.25)$$

The equation above represents the application of N Hadamard gates on N qubits, all initially in state $|0\rangle$; followed by the application of a diagonal gate $U(\theta)$ which is of the following form:

$$U(\theta) = \begin{bmatrix} e^{i\pi R(\theta_1)} & 0 & 0 & \dots \\ 0 & e^{i\pi R(\theta_2)} & 0 & \dots \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & e^{i\pi R(\theta_n)} \end{bmatrix} \quad (3.26)$$

where

$$R(\theta_k) = \begin{cases} 0 & \text{if } 0 \leq \theta_k < \pi \\ 1 & \text{if } \pi \leq \theta_k < 2\pi \end{cases} \quad (3.27)$$

Using (3.26) and (3.27) in (3.25), we have:

$$|\Psi(\theta)\rangle = \begin{bmatrix} e^{i\pi R(\theta_1)} \\ e^{i\pi R(\theta_2)} \\ \dots \\ e^{i\pi R(\theta_n)} \end{bmatrix} \quad (3.28)$$

$|\Psi(\theta)\rangle$ is therefore a vector whose terms belong to the set $\{1, -1\}$.

2. Note that when we defined the QUBO in equation (3.24), the variables were $y \in \{0, 1\}$. Since $|\Psi(\theta)\rangle \in \{1, -1\}$, we need to define our QUBO matrix using variables $z \in \{1, -1\}$. This new matrix is then called the spin-QUBO or sQUBO as defined in [58]. We shall denote the sQUBO matrix as Q' , which is a matrix of size $2n \times 2n$.

Let $z \in \{1, -1\}$ and $y \in \{0, 1\}$, then it is quite easy to verify that $y = \frac{1-z}{2}$. Using this in equation (3.22), we have:

$$C = \sum_k w_k \frac{1-z_k}{2} - P \left(\sum_{(i,j) \in E} \frac{1-z_i}{2} \frac{1-z_j}{2} \right) \quad (3.29)$$

In the search for optimal values of parameters z we can eliminate the constant terms in C as they only add a constant shift to the cost function. We can therefore aggregate the constant terms to an overall constant K . We can therefore simplify (3.29) as follows:

$$C = \sum_k w_k \frac{1-z_k}{2} - P \left(\sum_{(i,j) \in E} \frac{1-z_i}{2} \frac{1-z_j}{2} \right) \quad (3.30)$$

$$= \frac{1}{2} \sum_k w_k (1-z_k) - \frac{P}{4} \sum_{\substack{ij \\ i \neq j}} (1-z_i - z_j + z_i z_j) \quad (3.31)$$

$$= -\frac{1}{2} \sum_k w_k z_k + \frac{P}{4} \sum_{\substack{ij \\ i \neq j}} (z_i + z_j - z_i z_j) + K \quad (3.32)$$

The above *cannot* be represented as a QUBO matrix since it has linear terms which cannot be quadratized since $z_i^2 \neq z_i$.

Hence, we need to reformulate the problem. In this reformulation the linear terms are represented in the off-diagonal terms instead of the diagonals. Let us have $2n$ variables $\{z_1, z_2, \dots, z_{2n}\}$ where $z_1 \dots z_n \in \{1, -1\}$ and $z_{n+1} \dots z_{2n} \in \{1\}$. Then to represent a linear variable z_i , we can have the term $z_i z_{i+n}$ where $z_{i+n} = 1$. The equation (3.32) can be rewritten as follows:

$$C = -\frac{1}{2} \sum_k w_k z_k z_{k+n} + \frac{P}{4} \sum_{\substack{ij \\ i \neq j}} (z_i z_{i+n} + z_j z_{j+n} - z_i z_j) + K \quad (3.33)$$

This can now be written as:

$$C = z^T Q' z + K \quad (3.34)$$

This is the required Q' .

3. Equation (3.24) can have the following quantum equivalent:

$$C(\theta) = -\langle \Psi(\theta) | Q' | \Psi(\theta) \rangle \quad (3.35)$$

where $\theta = \{\theta_1 \dots \theta_n\}$ is a set of parameters and $|\Psi(\theta)\rangle$ is a parameterized ansatz that represents that vector y . The interesting thing about this representation is that we need only $\log n + 1$ qubits to represent Q' of size $2n \times 2n$. We put the negative sign as we want to make it a minimization problem.

4. Measure the expectation value (3.35) on a quantum computer or a simulator. In order to calculate this expectation value we need to decompose Q' into a sum of Pauli strings (see section 1.1.5).
5. Using a classical optimizer such as the genetic algorithm (GA), optimize the parameters θ .

$$C^*(\theta^*) = \min C(\theta) = -\sigma \tag{3.36}$$

From the optimized parameters we can get the binary solution using:

$$y_k = \frac{1 - \exp(i\pi R(\theta_k))}{2} = \begin{cases} 0 & \text{if } 0 \leq \theta_k < \pi \\ 1 & \text{if } \pi \leq \theta_k < 2\pi \end{cases} \tag{3.37}$$

3.2.5 Quantum-assisted algorithm to solve the Coordinator Problem

We now have a quantum model to solve the WP . We will call this the Quantum Worker Solver(QWS). To develop a quantum-assisted algorithm, the QWS is used together with a classical worker solver (CWS) to reach the optimal solution. The aim is to use the QWS together with the CWS where the CWS is used *only* when the QWS is not able to generate a new column.

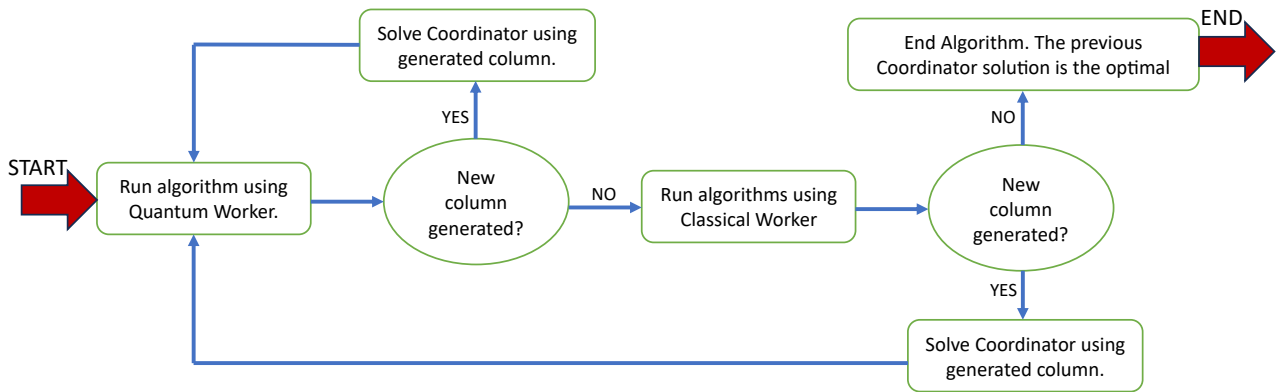


Figure 3.2: Progress of the quantum-assisted Algorithm

For every instance, we start the algorithm with the QWS as the worker. If the algorithm is able to generate a new column we use it to solve the coordinator (RCP) and continue with the optimization loop using the QWS. Note that generating a new column here means that the MWIS solution found gives a positive σ (as in equation (3.21)), hence violating (3.18). In case QWS is not able to generate a new column, we check whether the CWS can generate a new column. If the CWS is able the generate a new column, we solve the RCP using this new column and continue the optimization loop with the QWS. If even the CWS is unable to generate a new column, we have the optimal solution to the problem as the previous solution to the RCP obtained. This is explained diagrammatically in figure 3.2.

Using figure 3.2, we can modify Algorithm 4 to get Algorithm 5

Algorithm 5: Quantum-assisted Column Generation algorithm for RCP

Input: Incompatibility graph $G(K, E)$

- 1 Define R such that $R > \max \Gamma_I + 1$
- 2 $\Lambda' \leftarrow \emptyset$
- 3 **while** *True* **do**
- 4 Solve RCP and get primal dual couple (x', μ')
- 5 **for every model** v **do**
- 6 Solve WP using QWS
- 7 **if** $\sigma > 0$ **then**
- 8 Let the optimal solution be $I = (y_I, c_I)$.
- 9 $\Lambda' \leftarrow \Lambda' \cup \{I\}$.
- 10 **end**
- 11 **end**
- 12
- 13 **if** Λ' *has been modified* **then**
- 14 continue
- 15 **end**
- 16 **else**
- 17 **for every model** v **do**
- 18 Solve WP using CWS
- 19 **if** $\sigma > 0$ **then**
- 20 Let the optimal solution be $I = (y_I, c_I)$.
- 21 $\Lambda' \leftarrow \Lambda' \cup \{I\}$.
- 22 **end**
- 23 **end**
- 24 **if** Λ' *has been modified* **then**
- 25 continue
- 26 **end**
- 27 **else**
- 28 Current solution x' is the optimal solution
- 29 break
- 30 **end**
- 31 **end**
- 32 **end**
- 33
- 34 Apply rounding algorithm to transform the relaxed solution x' into a binary one.

3.3 Experimental Results

In this section, we demonstrate experiments carried out using tours of sizes 32 and 64. Synthetic data was generated with start and end times of tours as well as sets of allowed vehicles for every tour. We decide to have five different vehicle types and the cardinality of the set of allowed vehicles is 3. For simplicity, we do not generate location data. It can, however be added easily which will only change the density of the incompatibility graph. All runs of the quantum algorithm in this section are done using a quantum simulator.

In order to evaluate the contribution of the QWS, the number of successful quantum solves is compared to the total number of successful solves. A successful solve means that

Instance Size	Number of Qubits	Number of Instances	Mean % of Quantum Iterations
32	6	5	87.36
64	7	5	81.73

Table 3.3: Results for 32 and 64 Tours.

	Instance Size	
	32	64
Solver used for RCP	Gurobi	
Solver used for CWS	Gurobi	
Classical optimizer used for variational loop of QWS	Genetic algorithm (python package: geneticalgorithm)	
Penalty value	10	20
Average time taken to simulate a complete run of the algorithm on an instance	≈ 70 minutes	≈ 24 hours

Table 3.4: Supplementary information regarding the experiments

Genetic Algorithm Parameter	Instance Size	
	32	64
max_num_iteration	50	100
population_size	20	40
mutation_probability	0.1	
elit_ratio	0.05	
crossover_probability	0.5	
parents_portion	0.3	
crossover_type	uniform	
max_iteration_without_improv	None	

Table 3.5: Parameters of the Genetic Algorithm used in the experiments

Instance Size	No. of Qubits	GA Population Size	GA generations for QWS	Estimated Time (hours)
128	8	20	100	108
256	9	20	100	3500

Table 3.6: Estimated time for simulating instances of 128 and 256 Tours.

the QWS or CWS was able to generate a new column. In table 3.3, the number of quantum solves is shown as a percentage of the total number of solves, averaged over all the instances.

Tables 3.4 and 3.5 give some important details regarding the experiments.

Figure 3.3 shows in detail complete runs of algorithm 5 for instances of size 32 and 64. Each line represents an instance. We have 5 instances. For every instance, the first point of a plot is a gray square. This is a quick greedy solution to the problem and is the starting point of our algorithm. This greedy algorithm colors each node one by one, ensuring that

neighboring nodes do not share the same color. This ensures that we already have a decent solution to start with. As we go forward in the plots, we have blue diamonds and red pentagons which signify the RCP solutions obtained using the QWS and the CWS respectively. Finally we have the black hexagon which is the optimal point, where both the QWS and CWS were not able generate a new column. The cost function has been normalized using the formula :
$$\frac{(\text{Current Solution}) - (\text{Optimal LP-feasible Solution})}{\text{Optimal LP-feasible Solution}}$$
.

The iterations of the algorithm are different from and not to be confused with the GA generations of the QWS. The name *max_num_iteration* (as used in Tables 3.4 and 3.5) is the parameter name for the number of generations in the python package.

Taking the instance with the blue dash-dotted line for 64 tours as an example, there are 16 iterations where the QWS was successful, 1 where QWS was unsuccessful and CWS was successful and 1 where both QWS and CWS were unsuccessful. Note, however, that for every iteration we called the QWS. Hence, there were a total of 18 QWS calls. Every QWS call requires 5 MWIS solutions (one for each vehicle type) each requiring around 300 expectation value measurements, hence the total number of expectation value measurements required were of the order of 10^4 . Even though the size was tractable to represent our problem, it was therefore not feasible to run the algorithm on a quantum computer since every expectation value measurement on an IBM quantum computer can take from a few seconds to a few hours, depending upon the wait time (or queue time). In addition we were limited to a size of 64 nodes for simulation since for 128 nodes and above, the simulation time was too long to carry out experiments. We can see in Table 3.6 the estimated time that might be required to potentially simulate instances of 128 and 256 tours. These estimates are given considering the population-size and number of GA generations to be 20 and 100 respectively, and considering that there are 25 algorithm iterations to converge to a solution. Here, the QPU should have significant a time advantage over the simulators.

The algorithm is a heuristic and therefore might not have the same performance on every run. This is shown in figure 3.4, where one specific instance of size 32 is run 5 times.

Finally, figures 3.5 and 3.6 show the result of a 32-tour instance. This is specifically the instance denoted by a black solid line in figure 3.3. Figure 3.5 shows the incompatibility graph and figure 3.6 shows the corresponding time-window diagram. The nodes in figure 3.5 do not refer of any specific location. The tour numbers from 0 to 31 are marked on the time-window bars as well as the nodes. The 5 vehicles used to carry out all the tours are marked using different colors.

3.4 Extension: Replacing CWS with better performing QWS

In the previous section, we have seen how the CWS is used when the QWS is not able to generate a new column. The CWS therefore acts as a secondary check. An extension to the algorithm can be to replace the CWS with a better performing QWS. We will call this extension full-quantum to differentiate it from the hybrid quantum-assisted method described before.

We have seen in Table 2.10 that the performance of the LogQ algorithm improves with an increase in the number of GA generations. This can be further seen for a 64-tour fleet conversion instance in figure 3.7. The run with 100 GA generations fails at iteration 7 and requires the help of CWS to move forward. We can see that even iterations 8 and 9 require the help of CWS to improve the cost function. For the run with 200 GA generations, the algorithm is stuck at the same value of cost function (5250) and requires the help of CWS

in iteration 5. However, in iteration 6 and 7 we can see that the algorithm could find an improvement in the cost function without the help of the CWS.

We therefore propose an extension to the algorithm where the CWS is replaced by a QWS with double the number of GA generations. This means that when the QWS will fail, instead of running the classical Gurobi solver to check for improvements, we run the QWS again, this time with double the number of GA generations, and therefore, a higher chance to overcome a local minima if it the algorithm is stuck in one.

While this of course does not guarantee an optimal solution, it is a interesting strategy if one does not wish to use the CWS. Figure 3.8 shows the results for a 64-Tour instance with the CWS replaced by QWS with twice the number of GA generations. The results are superposed on the algorithm run where the CWS was used.

The 'Best Solution found with Full-Quantum' in the figure refers to the point where both the QWS with 100 generations (QWS100) and the QWS with 200 generations (QWS200) fail to generate a new column. In the hybrid quantum-assisted run (the blue line), QWS100 fails in iteration 17 and the CWS is required to improve the cost function in iteration 18. In the full-quantum run (the black line) the QWS100 fails in iteration 14 but QWS200 improves the cost function in iteration 15. In this way, it can be seen that a QWS with higher number of GA iterations can be used to help the algorithm if it is stuck in a local minima.

Thus, a combination of different GA generations can be used to further generalize this technique of using secondary checks.

3.5 Discussion and Conclusion

In this chapter we have successfully demonstrated a quantum-assisted algorithm to solve the Fleet Conversion problem. This shows firstly the advantage of having an algorithm that scales logarithmically with the size of the problem. This trait of LogQ helped us model problems of sizes that are well outside the realm of what other quantum variational algorithms can handle. Notably, our algorithm successfully handles instances that, albeit synthetic, represent realistic, non-trivial industrial problem sizes, which displays the potential utility of leveraging quantum computing for solving complex optimization problems.

Secondly, it demonstrates the possibility of harnessing quantum algorithms in conjunction with classical solvers rather than being in competition with them. If the problem is very resource intensive then instead of the using a full classical optimization, a part of the computation can be transferred to the quantum computer, potentially reducing required computational resources (logarithmic representation).

There remain, however, several drawbacks in the algorithm. Firstly, the algorithm requires the decomposition of the Hamiltonian matrix into a sum of Pauli strings. The cost of doing this using the method used here (as shown in section 1.1.5) increases significantly with an increase in the number of qubits. Reducing the time to decompose the matrix could reduce runtime by a large margin for larger instances.

Secondly, larger instances require a larger number of runs on the genetic algorithm and therefore it quickly becomes infeasible to increase problem size. Even if we decide not to increase the number of genetic algorithm runs, the quality of the QWS solution will degrade resulting in more QWS calls, hence a higher number of iterations (considering we require the same level of performance).

The success of the algorithm depends on the accuracy of the QWS solution. Worse QWS solutions inadvertently lead to more QWS calls. Hence the runtime of the algorithm could be quite sensitive to noise in potential runs on a quantum computer. The indicates that this

algorithm would be suitable for small but fault tolerant quantum computers rather than larger ones with higher levels of noise. Despite the fact that we are moving towards a large number of physical qubits in the future, the availability of logical (fault-tolerant) qubits will remain limited. Consequently, this approach will continue to be relevant.

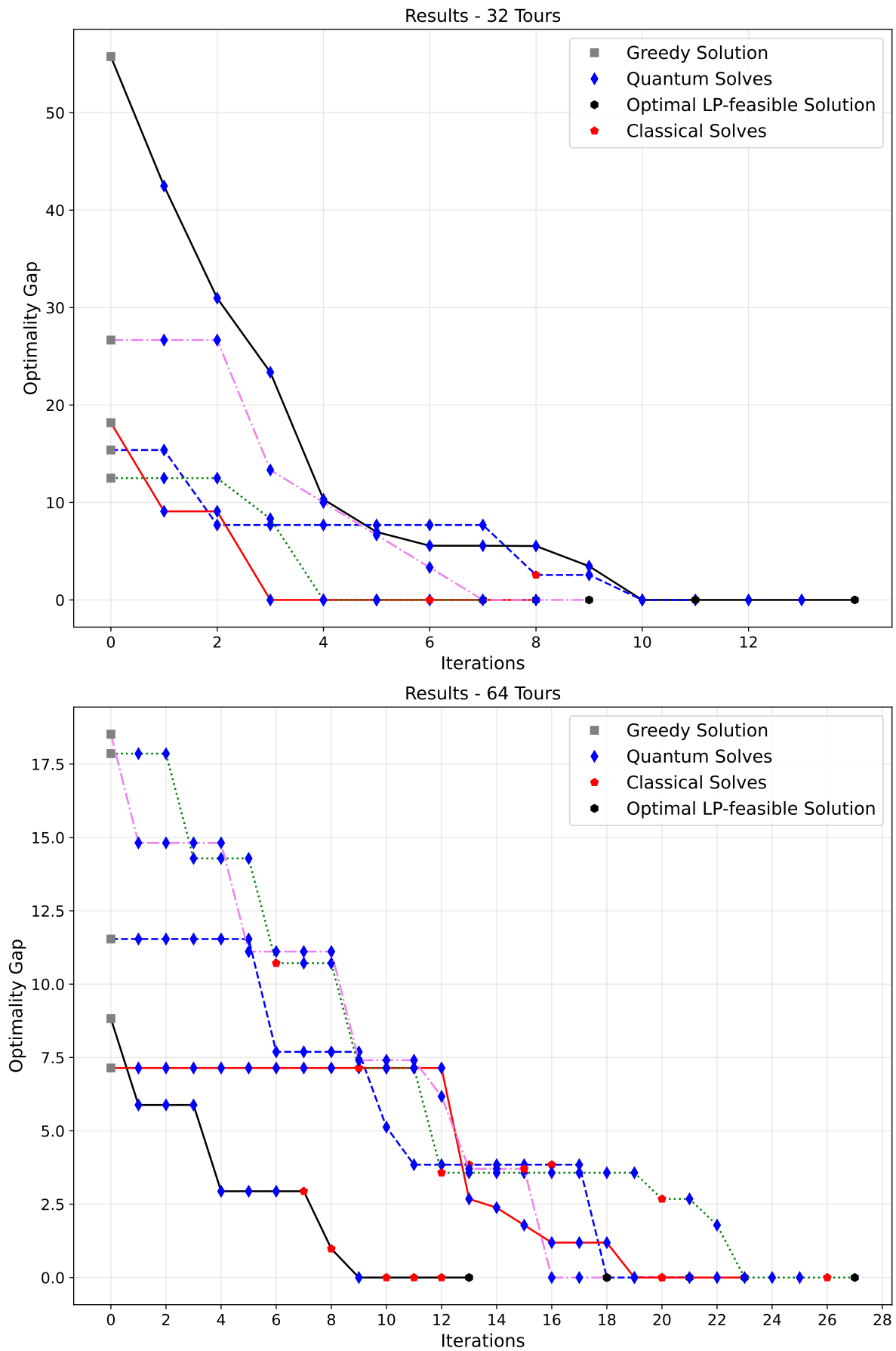


Figure 3.3: Full run of the algorithm for instances of 32 and 64 tours.

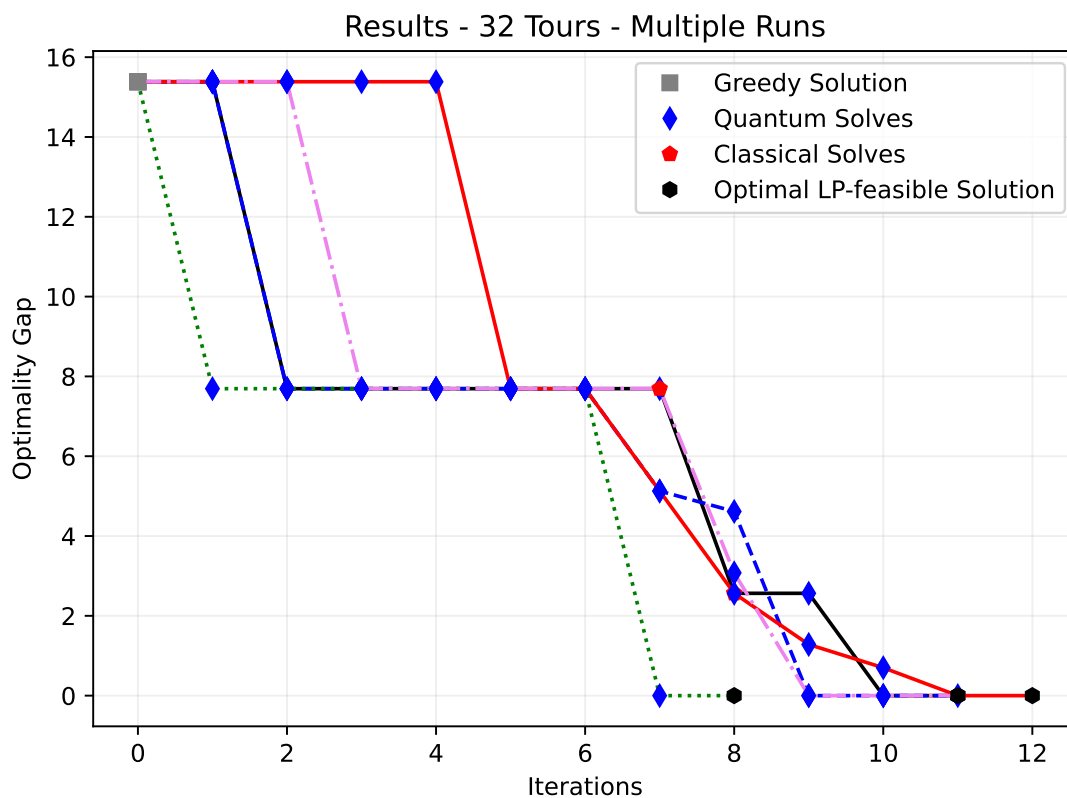


Figure 3.4: 5 full runs of the algorithm for an instance of size 32.

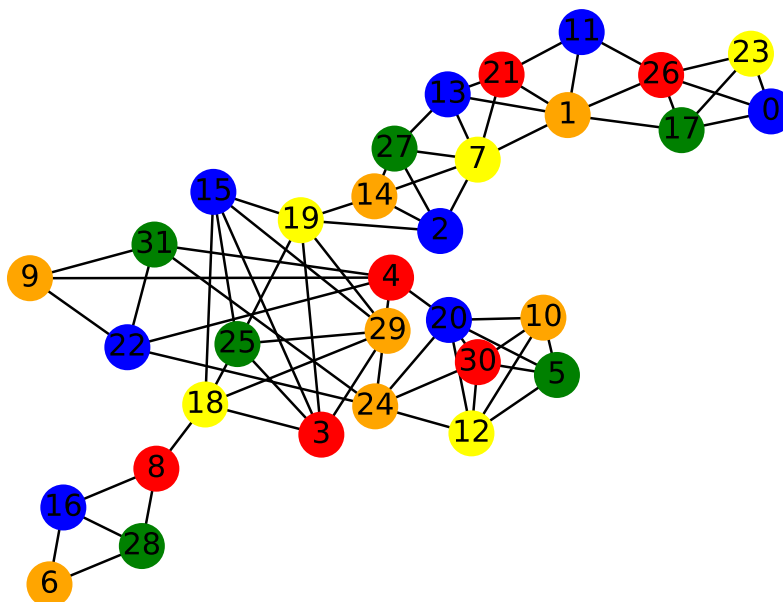


Figure 3.5: Incompatibility graph for a 32-tour instance.

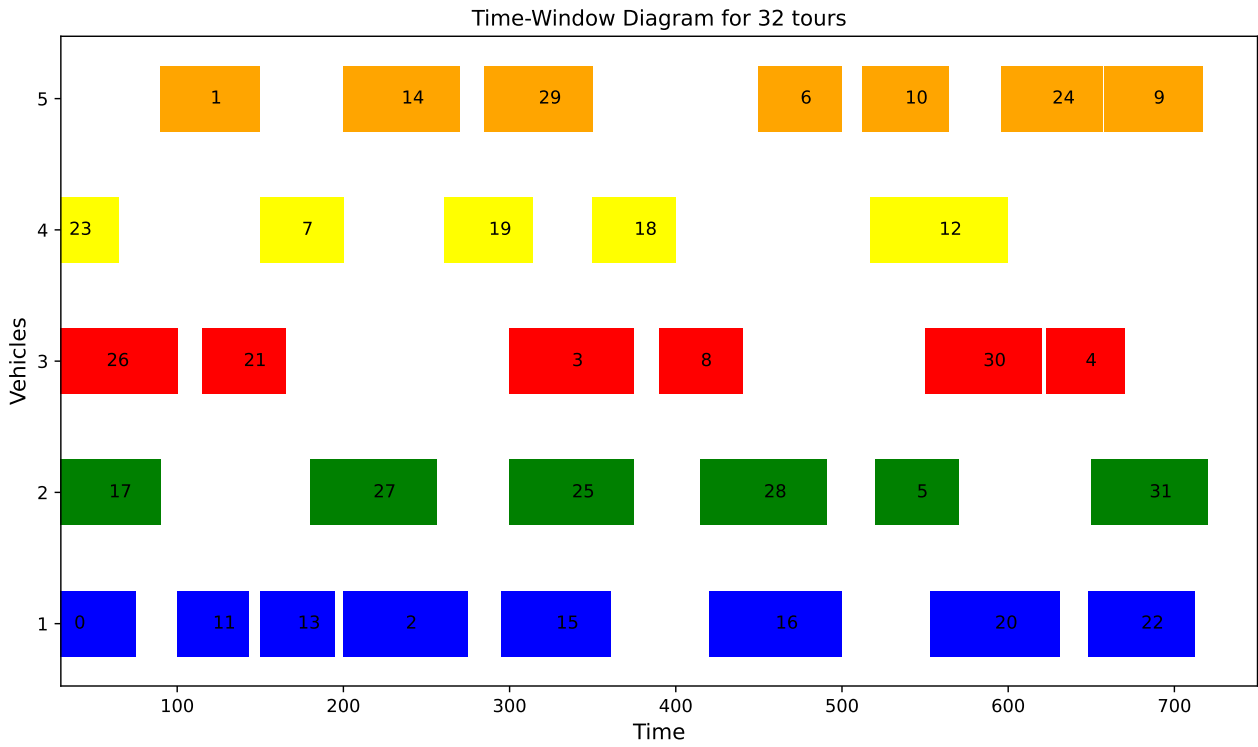


Figure 3.6: Time-window diagram for a 32-tour instance.

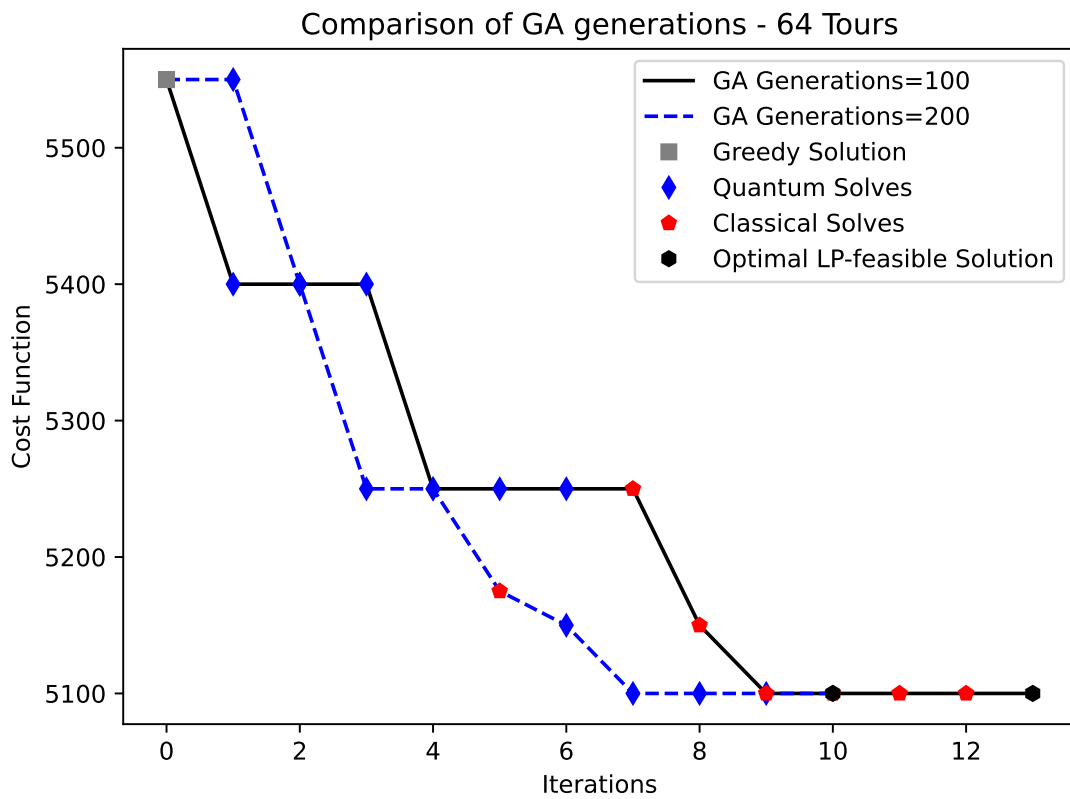


Figure 3.7: Performance comparison between 100 and 200 GA generations for a 64-Tour Instance.

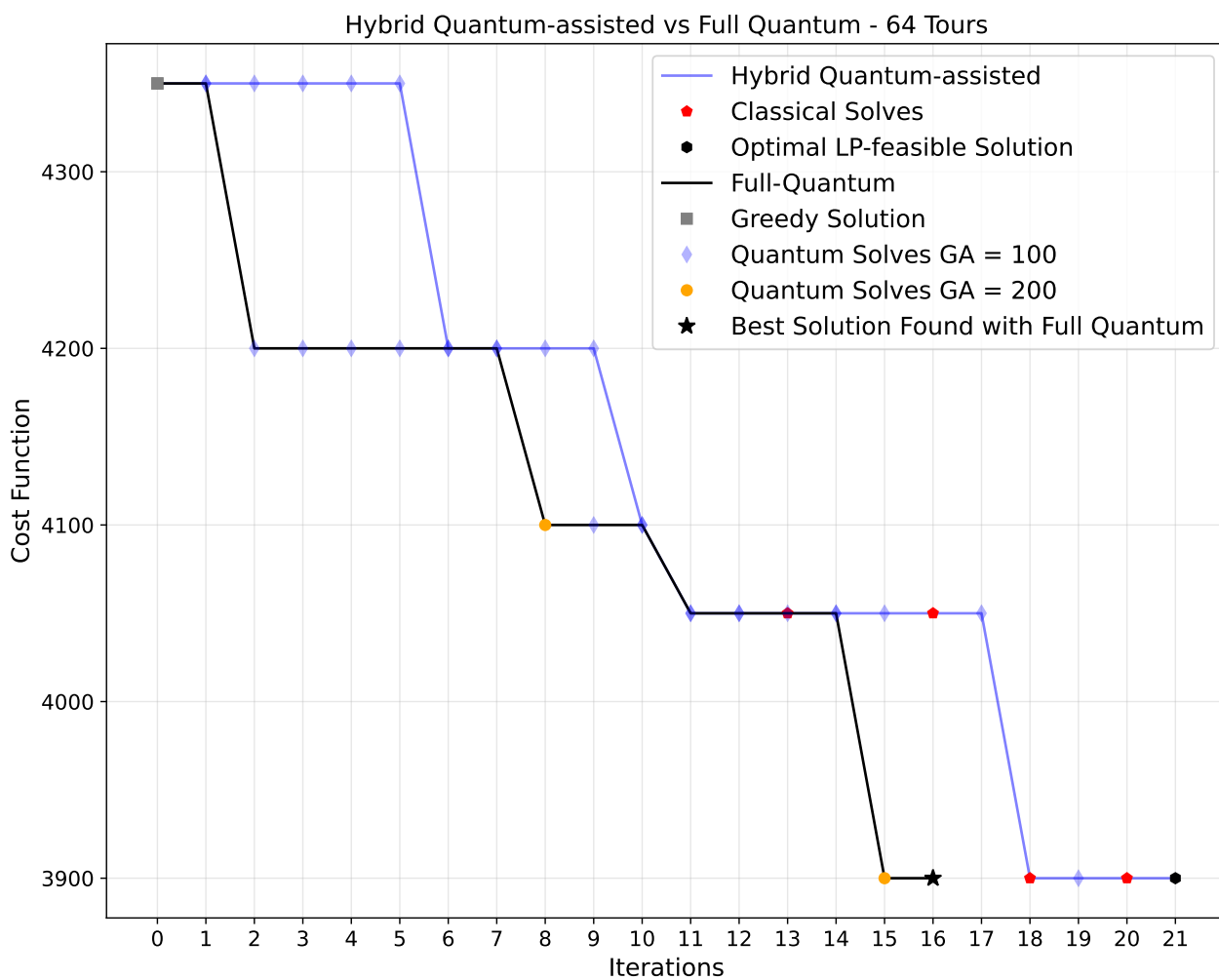


Figure 3.8: Comparison between the algorithm runs of the hybrid quantum-assisted vs full-quantum versions - 64 Tours.

Mesh Segmentation using the LogQ encoding

In the previous chapter, we saw the application of the LogQ encoding for the problem of fleet conversion, a problem in the domain of transportation. In this chapter, we focus on another use-case of the LogQ encoding in the domain of computer modeling. This chapter is based on work done in the context of a deliverable [81] for the Horizon 2020 NEASQC project.

4.1 Background and Context of the Project

The work presented in this chapter represents our contribution to the Horizon 2020 project NExt ApplicationS of Quantum Computing or $\langle \text{NE}|\text{AS}|\text{QC} \rangle$, funded by the European Union. This project is aimed at investigating and developing applications of quantum computing, especially with Noisy Intermediate Scale Quantum (NISQ) computers. It is specifically geared towards practical use-cases for industrial end-users.

The NEASQC project, with a budget of over 4 million euros, unites several academic institutions as well as private companies including TotalEnergies. In order to showcase the potential of near-term quantum computers, nine use cases from various fields have been identified. These use-cases, divided into 3 axes, are as follows:

1. Symbolic AI and Graph Algorithmics
 - (a) Quantum Probabilistic Safety Assessment (QPSA)
 - (b) Quantum Natural Language Processing (QNLP)
 - (c) Quantum rule-based systems (QRBS) for breast-cancer detection
2. Machine Learning and Optimization
 - (a) **HPC Mesh Segmentation**
 - (b) Financial Applications
 - (c) Reinforcement learning for inventory management
 - (d) Hard optimization problems for smart charging of electric vehicles
3. Chemistry

- (a) Drug Discovery
- (b) CO₂ Recapture

The use-case 2(a), HPC Mesh Segmentation, was carried out at TotalEnergies. There were two deliverables associated with this use-case. They were:

1. Benchmarking of a quantum algorithm for mesh segmentation against classical mesh segmentation techniques such as KMeans.
2. Software package for mesh segmentation tasks.

This chapter will present the work done with respect to the first deliverable. It was reviewed by three external members of the NEASQC consortium and made available on the NEASQC website in April 2024 [81]. The software package, which combines codes of the LogQ encoding as well as the implementation of the use-case, forms the second deliverable and is to be delivered in October 2024.

4.2 Motivation

Nowadays, computer modeling plays a crucial role in the core domains of TotalEnergies' activities such as Earth imaging, physical modeling for reservoirs, and others. Consequently, TotalEnergies is heavily investing in the development of new approaches and techniques which could improve the quality and efficiency of the corresponding software. While some efforts of the R&D group target independent improvements of software and algorithms, some limitations are more global and concern the basic principles of numerical modeling. One of these limitations is related to the discretized representation of the objects used in computer modeling, e.g., the mesh. Before starting any realistic simulation, the numerical mesh, which corresponds to a collection of edges, faces, and connecting points, has to be pre-processed. This pre-processing, which largely simplifies further treatment, includes an essential stage of dividing the mesh into meaningful parts. This process of splitting a numerical mesh is called *Mesh Segmentation*. In this chapter we tackle the use-case of Mesh Segmentation using the LogQ encoding.

One of the simplest types of meshes are triangular meshes. A triangular mesh consists of a collection of triangles that together form a surface or solid representation of a 2D or 3D object. Each triangle is defined by its three vertices, and the entire mesh is formed by connecting these vertices. Each node of the mesh has an associated color or property. In this chapter we will use only 2D triangular meshes to simplify the representations. 3D mesh problems, however, can follow the same procedure. In order to solve the problem of mesh segmentation using our algorithm, we first need to convert it into a graph optimization problem consisting of an objective function and constraints if necessary.

To do this we create a graph from the mesh. The vertices of the graph are the nodes of the mesh and the edges of the graph are the sides of the triangle. The edges are then weighted with a weight equal to the difference in color between the vertices. Hence we have an edge-weighted graph.

Once we have a graph, we need to define an objective function that will help us get the different segments of the mesh. Here we will use the maximization of modularity [82] for meshing. Modularity is a measure of how well a network is split into different groups or clusters. It helps us see if these groups have more connections within themselves than we would expect just by random chance. We define the modularity objective function in

the form of a QUBO problem and then use the LogQ encoding based algorithm to get the solution.

This chapter is organized as follows. We start with describing the problem of mesh segmentation in section 4.3, followed by its modelization as a graph problem in section 4.4. We then explain in section 4.5 how we can define an objective function in order to perform mesh segmentation from a given graph. In section 4.6 we provide an extension to definition of the objective function followed by section 4.7 where we define the benchmarking metrics to be used in the chapter. We show the benchmarking results of our algorithm with respect to the KMeans algorithm in section 4.8 and finally in section 4.9 we have the conclusion.

4.3 Mesh Segmentation

A mesh is a representation of a surface or object formed by connecting vertices, edges, and faces. Each face of the mesh is defined by the vertices and the edges connecting them. Meshes are widely used in computer graphics and simulations because they provide a simple and efficient way to approximate complex surfaces. The vertices store positional information (x, y, z coordinates), edges connect pairs of vertices, and faces define the surface geometry. Figure 4.1 (taken from [83]) gives an example of an object and its corresponding meshes. There are two meshes of the object shown in the example, one being a moderate mesh and the other a coarse mesh. The coarse mesh incorporates a higher level of approximation.

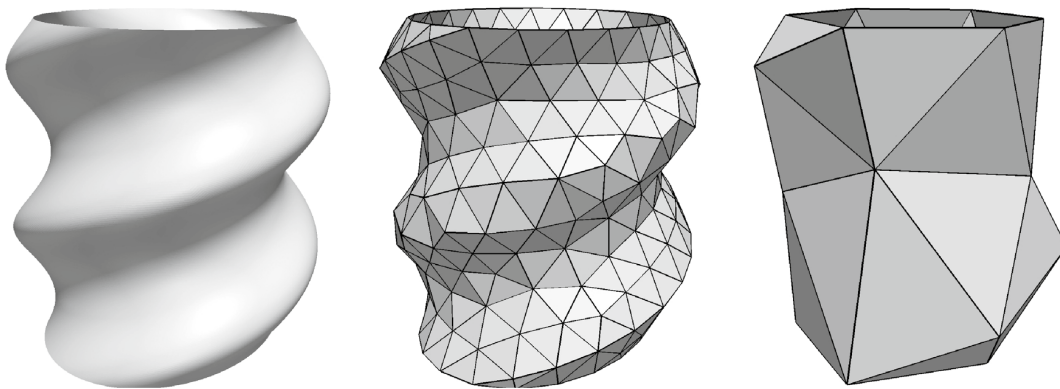


Figure 4.1: An example of 3D meshing of an object.

Mesh segmentation refers to the process of dividing a complex mesh (composed of vertices, edges, and faces) into meaningful and semantically coherent parts or regions. The goal is to identify and isolate distinct components within the mesh, which may correspond to different objects, substructures, or functional units. Take for example figure 4.2 where a 3D geological surface is represented as a triangular mesh. It is then segmented into various subdomains depending on specific properties.

Meshes can be in various types, such as triangle meshes, quadrilateral meshes, or polygonal meshes, depending on the type of polygons used to construct the faces. In this chapter we will deal with only 2D triangular meshes. A 2D mesh can be seen as a cross section of a 3D mesh.

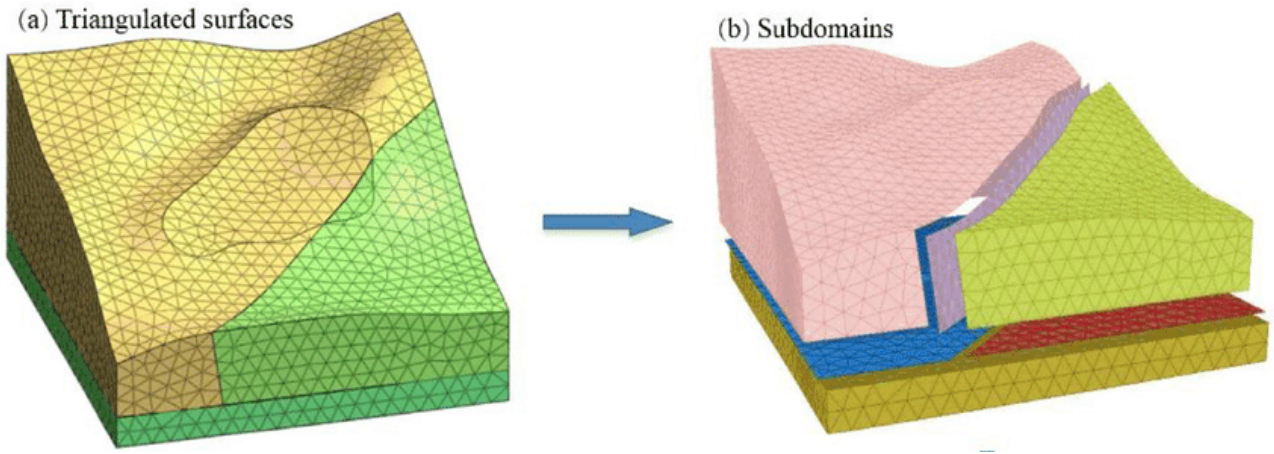


Figure 4.2: An example for the segmentation of a geological model represented as a triangular mesh (taken from [84]).

4.4 Mesh Segmentation as a graph optimization problem

In order to tackle the problem of mesh segmentation, we first convert it into a graph optimization problem. Figure 4.3 shows a 2D triangular mesh. It has 64 vertices and every vertex has a color (or property) associated with it. The color is a number ranging from 10 to 110. In order to translate this problem of mesh segmentation into a graph problem, we create a graph of 64 vertices having the same edges as the edges of the triangular mesh, as shown in Figure 4.4. Then the edges of the graph are assigned weights equal to the difference in color between the vertices of the edge. The goal is to divide the graph into clusters of vertices. The number of segments or clusters required needs to be specified.

4.5 Description of the Objective Function

Once the mesh is represented as an edge-weighted graph, we need to define an objective function to divide it into segments. Here we will use the maximization of modularity [82] as our objective.

Modularity is a measure to evaluate the quality of a partition of a network into communities or clusters (also called communities or groups). The idea of maximizing the modularity is to maximize the number of connections between the vertices within clusters while having sparse connections between vertices in different clusters.

Mathematical definition of Modularity

Consider an edge-weighted graph $G(V, E, w)$ with weights $w_{ij}, i, j \in V$. Let the variable $x_i \in \{1, -1\}$ decide whether the vertex i is in one subgroup or the other. Then its modularity to maximize for division into 2 subgroups defined as follows :

$$M = \frac{1}{4m} \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(x_i, x_j) \quad (4.1)$$

where:

1. k : weighted degree of vertex i

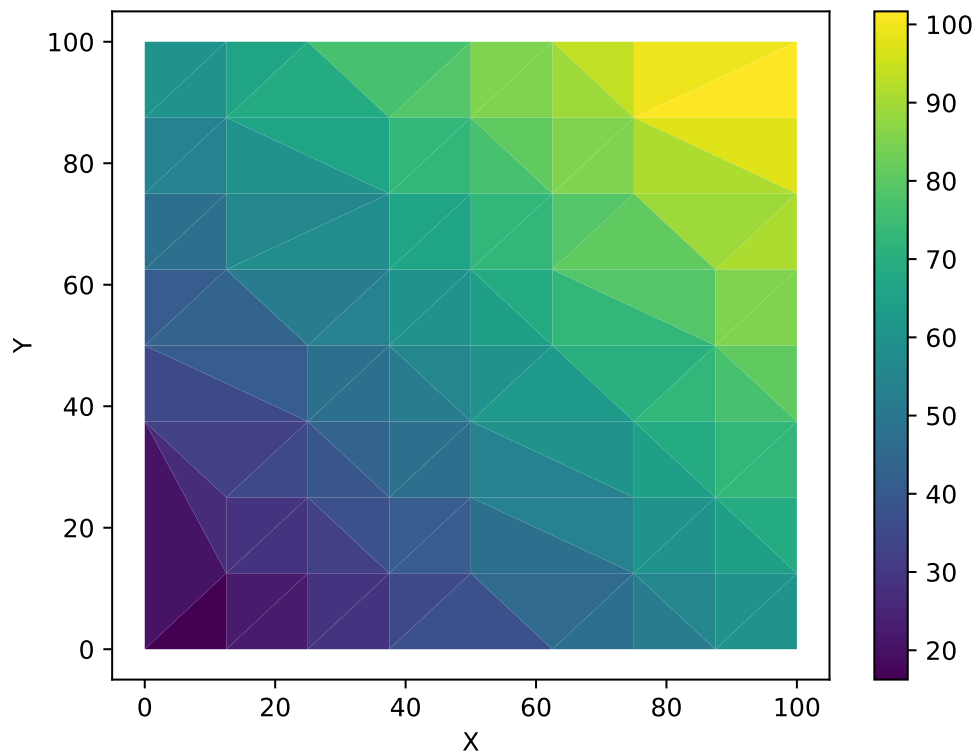


Figure 4.3: 2D Triangular Mesh of Size 64

2. A_{ij} : adjacency matrix of G

$$A_{ij} = \begin{cases} w_{ij} & \text{if } (i, j) \in E \\ 0 & \text{if } (i, j) \notin E \end{cases} \quad (4.2)$$

3. $2m$ is the sum of the weights of all the vertices.

$$m = \frac{1}{2} \sum_{i \in V} k_i \quad (4.3)$$

4. $\delta(x_i, x_j)$ is the Kronecker delta function. It is 1 if both vertices are in the same cluster and 0 if they are in different clusters.

$$\delta(x_i, x_j) = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{if } x_i \neq x_j \end{cases} \quad (4.4)$$

Since $x_i \in \{1, -1\}$, it can be modeled as:

$$\delta(x_i, x_j) = 1 + x_i x_j \quad (4.5)$$

Hence,

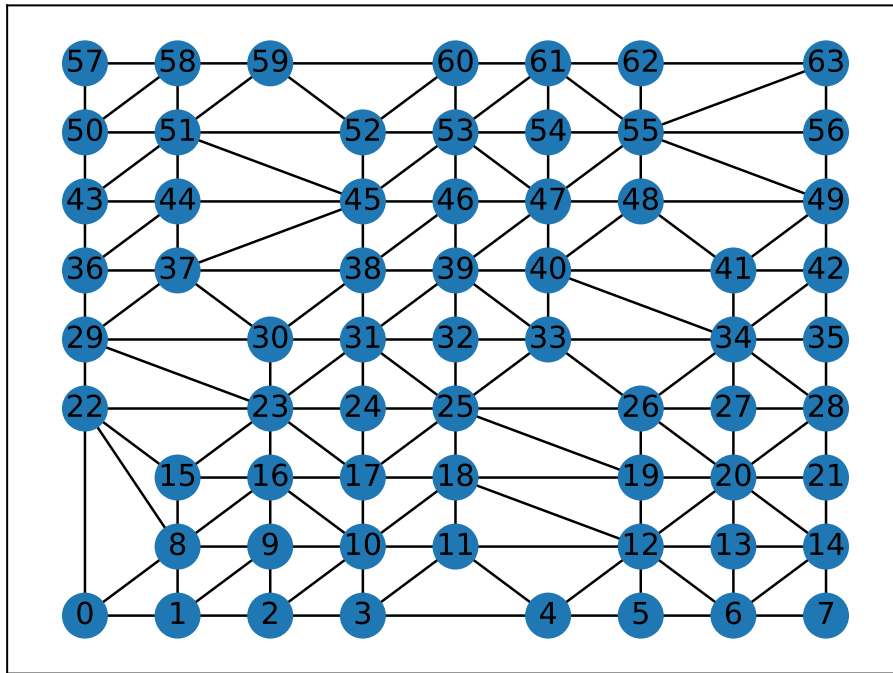


Figure 4.4: 2D Triangular Mesh of Size 64 as a Graph Problem

$$M = \frac{1}{4m} \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) (1 + x_i x_j) \quad (4.6)$$

We can drop the constant factor in the above equation as it does not affect the optimization process. Therefore we have the following expression for the objective function.

$$M = \frac{1}{4m} \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) x_i x_j \quad (4.7)$$

Understanding the components of Modularity

For any two vertices, the modularity objective function aims to maximize the weight between them if they are in the same cluster. All the components of the modularity function are explained in detail below.

- The adjacency matrix A_{ij} tells us the weight of the edge between the vertices i and j . $A_{ij} = w_{ij}$ if there is an edge between the vertices i and j , else $A_{ij} = 0$.
- $\frac{k_i k_j}{2m}$ is the *expected weight* of the edges between vertices i and j for a random distribution of the vertices.
- $A_{ij} - \frac{k_i k_j}{2m}$ therefore represents the difference between the actual weight between i and j and the expected weight between them in case of a random assignment

of vertices. A higher value of $A_{ij} - \frac{k_i k_j}{2m}$ would mean that there is more weight between 2 edges than would be expected by chance.

- (d) Finally we have the delta function that defines whether 2 vertices are in the same cluster or not. Maximizing the product $A_{ij} - \frac{k_i k_j}{2m} \delta(x_i, x_j)$ maximizes the weights between the nodes when they are in the same cluster, which is the goal of clustering.

Modularity as a QUBO

In order to be compatible with the LogQ encoding, modularity needs to be represented in the form of a QUBO. Equation (4.7) is already in the form of a QUBO and can be written as :

$$M = \frac{1}{4m} x^T Q x \quad (4.8)$$

$$Q_{ij} = A_{ij} - \frac{k_i k_j}{2m} \quad (4.9)$$

Q is the required QUBO matrix that can be used in LogQ.

In order to get more than two divisions, we use the algorithm repeatedly on the graph. This method limits us to number of clusters which are a power of 2. The numerical results of this chapter are only based on this method. The algorithm can however, be extended to a general number of clusters as well by the generalization of the definition of modularity as explained in the next section.

4.6 Modularity for more than two segments

The definition of modularity given above can generate only two segments and in order to get multiple segments the process of bi-partition is done repeatedly. This method, however, might not be the optimal method to get the best segments. Moreover, we are limited to only sizes which are a power of 2. Therefore, it is important to generalize the objective function for any number of segments.

Let there be c segments and let $x_{ip} \in \{1, 0\}$ be the binary variable such that:

$$x_{ip} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to the segment } p \\ 0 & \text{otherwise} \end{cases} \quad (4.10)$$

Then we can redefine modularity for c segments as:

$$M = \frac{1}{4m} \sum_{p=1}^c \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) x_{ip} x_{jp} \quad (4.11)$$

The delta function used in the previous formulation is replaced by $x_{ip} x_{jp}$. This makes sure that the only non zero terms are when both $x_{ip} = x_{jp} = 1$, or in other words when both vertices i and j are in cluster p .

In addition we need to add constraints to make sure that every vertex is only assigned to a single cluster. These constraints can be written as:

$$\sum_{p=1}^c x_{ip} = 1, \quad \forall i \in V \quad (4.12)$$

Unlike the case of bi-partition of the clusters, the optimization problem in this case is a constrained one. Therefore, we need to include the constraints as penalty terms in the objective function. The QUBO cost function to maximize therefore becomes:

$$\mathcal{C} = \frac{1}{4m} \sum_{p=1}^c \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) x_{ip} x_{jp} - P \left(\sum_{p=1}^c x_{ip} - 1 \right)^2 \quad (4.13)$$

P is a real number that denotes the strength of the penalty. It is generally set to be more than the maximum value of the objective function (without the constraints). In this case that is $P = \sum_{i,j \in V} w_{ij}$.

In addition, the problem has been formulated using variables $x \in \{0, 1\}$ and therefore, using the method demonstrated in section 2.2.3, we need to convert from the QUBO to sQUBO (QUBO with variables $x \in \{1, -1\}$).

For a mesh of size n , in order to divide it into c segments, the size of the QUBO matrix will be $nc \times nc$ and subsequently the size of the sQUBO will be $2nc \times 2nc$. Therefore $\lceil \log_2 2nc \rceil$ qubits are required to run the algorithm, as opposed to the bi-partition case where $\lceil \log_2 n \rceil$ qubits are required. As mentioned before, only the bi-partition case is used to generate the results of this chapter.

4.7 Metrics to evaluate the quality of the segments

In order to benchmark the LogQ-encoding based algorithm, we will use two metrics: rand index and cut discrepancy. In both the measures, there is a *predicted* segmentation and a *ground truth* segmentation. The ground truth segmentation is the result of the method used as benchmark and the predicted segmentation is the result of the LogQ-based algorithm. In this chapter the KMeans clustering algorithm [85] is used as the benchmark. The true and predicted segmentations are lists of segment labels. Here we present the mathematical description of the measures. The definitions are adapted from [86].

4.7.1 Rand Index

The Rand index is measure of similarity between two segmentations given by W.M.Rand [87]. Given two segmentations, it evaluates the likelihood of a pair of nodes being either in the same segment in both segmentations, or in different segments in both segmentations.

Let S_1 and S_2 be the two segmentations, s_i^1 and s_i^2 denote the segment label of node i in S_1 and S_2 and N be the number of nodes of the mesh. In addition let:

$$C_{ij} = \begin{cases} 1 & \text{if } s_i^1 = s_j^1 \\ 0 & \text{otherwise} \end{cases} \quad (4.14)$$

$$P_{ij} = \begin{cases} 1 & \text{if } s_i^2 = s_j^2 \\ 0 & \text{otherwise} \end{cases} \quad (4.15)$$

Rand Index (RI) is then defined as:

$$RI(S_1, S_2) = \binom{2}{N}^{-1} \sum_{\substack{i,j \\ i \leq j}} C_{ij} P_{ij} + (1 - C_{ij})(1 - P_{ij}) \quad (4.16)$$

where

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} \quad (4.17)$$

4.7.2 Cut Discrepancy

Cut Discrepancy [88] is a metric used to compare the boundaries of two segmentations. It does this by measuring how close the boundaries of one segmentation are to the boundaries of another segmentation. Specifically, it sums up the distances from points on the predicted segmentation boundaries to the nearest points on the ground truth boundaries, and vice versa. This provides a measure of how similar the boundary placements are between the two segmentations.

Let C_1 and C_2 be the sets of points on the segment boundaries of segmentations S_1 and S_2 . Let $d_G(p_1, p_2)$ be the distance between the points $p_1 \in C_1$ and $p_2 \in C_2$. The distance between $p_1 \in C_1$ and the set C_2 is defined as:

$$d_G(p_1, C_2) = \min(d_G(p_1, p_2), \forall p_2 \in C_2) \quad (4.18)$$

Directional Cut Discrepancy of S_1 with respect to S_2 is defined as:

$$DCD(S_1 \rightarrow S_2) = \text{mean}(d_G(p_1, C_2), \forall p_1 \in C_1) \quad (4.19)$$

Finally, the cut discrepancy (CD) is defined as:

$$CD(S_1, S_2) = \frac{DCD(S_1 \rightarrow S_2) + DCD(S_2 \rightarrow S_1)}{\text{avgRadius}} \quad (4.20)$$

Where *avgRadius* is the average distance from a point on the mesh boundary to centroid of the mesh.

4.8 Results

In this section we show the results obtained for meshes of sizes 32, 64 and 128. The meshes are generated synthetically using packages *numpy* and *matplotlib* on Python. Results of the KMeans clustering method are used as a benchmark. KMeans clustering is also an NP-Hard problem and therefore has exponential complexity. The KMeans algorithm of the *scikit-learn* Python package is used here. This package uses Lloyd's algorithm [89] as the default algorithm and this is what was used here. The worst case complexity is given by $O(n^{\frac{k+2}{p}})$ where n is the number of vertices, k is the number of segments and p is the number of attributes. Here we have only one attribute for every node (color) so $p = 1$.

The quantum simulations as well as the classical algorithm runs are carried out using the following computational resources: Chip: Apple M2 Pro, RAM: 32 GB. For the LogQ runs, the meshes of sizes 32, 64 and 128 required 5, 6 and 7 qubits respectively. The genetic algorithm is used as the classical optimizer for the variational loop in LogQ. The parameters of the genetic algorithm such as the *population size* and the *maximum number of iterations* have an impact on the quality of the solutions and the execution time. In order to choose these parameters, one must consider a balance between execution time and the quality of solution. Increasing the population size and number of iterations will improve the solution up to a certain point after which it will stabilize. Conversely, augmenting these parameters also leads to a linear increase in execution time. In our case, both the population size and maximum number of iterations are set to 40.

In figures 4.5 and 4.6 we can visualize the mesh segmentation results for the instance of size 128 for 8 segments, using KMeans and the LogQ algorithm respectively. We can see that in KMeans it is perfectly separated into 8 different clusters while for the quantum algorithm

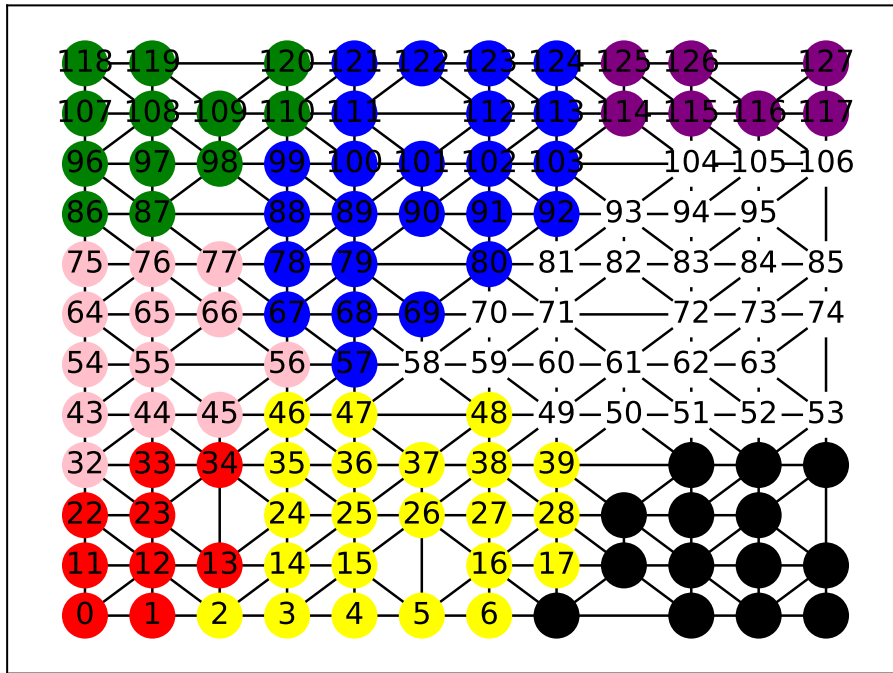


Figure 4.5: Mesh Segmentation of 128-Node Mesh into 8 segments using KMeans

it is less consistent, with some of the clusters being a bit scattered. This might be due to the repeated bi-partitioning and sometimes the algorithm has no choice but to split the clusters.

The benchmarks shown in tables [4.1](#), [4.2](#) and [4.3](#) are generated by comparing the segments generated by the LogQ algorithm and the segments generated by KMeans. For every instance, we generate the segments using both LogQ and KMeans. Then, we consider the KMeans solution to be the true segmentation and the LogQ solution to be the predicted segmentation. Note that KMeans does not necessarily provide the optimal solution, but we consider it as a classical benchmark.

For cut discrepancy, smaller values are better while for rand index, larger is better. The Rand index ranges from 0 to 1, where 1 indicates perfect agreement between the two clusters, and 0 indicates no agreement beyond what would be expected by random chance. Cut discrepancy is the difference in normalized cuts between the two clusters. It varies from 0 to a positive number, 0 being the best.

Cut discrepancy depends heavily on the accuracy of boundaries. This is why as the number of segments increases, it becomes harder to maintain a low cut discrepancy as the precision of the boundary recognition plays an important role. For all 3 mesh sizes, the cut discrepancy is close to 0 for $k = 2$ and gets significantly worse as we increase the number of segments. This indicates that both KMeans and LogQ find a very similar boundary for $k = 2$ and then the boundaries of the segments grow more and more different.

Rand index is actually getting better with higher number of segments. Therefore with increasing number of segments while segment boundaries are not as consistent, as demonstrated by the cut discrepancy, the number of pairs belonging to the same segment increases.

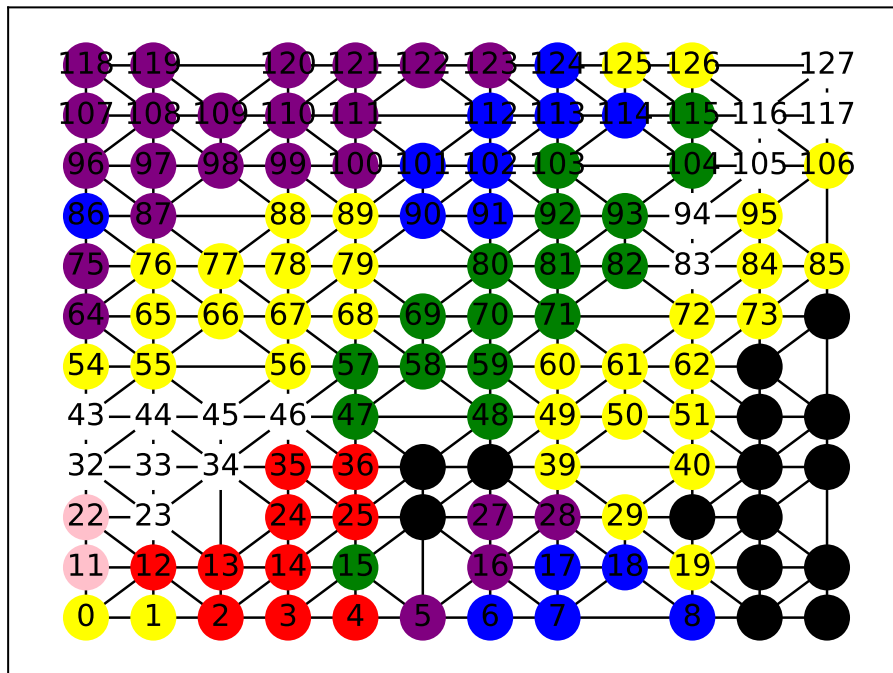


Figure 4.6: Mesh Segmentation of 128-Node Mesh into 8 segments using LogQ

The main takeaway from the benchmarks is that the method works well for bi-partition but the segment boundaries get worse with an increase in k . The results are quite disappointing and further work is therefore required to improve the consistency of the segment boundaries as the number of segments increase.

Number of Segments	Cut Discrepancy	Rand Index
2	0.004	0.22
4	0.08	0.4
8	0.77	0.62

Table 4.1: 32-Node Mesh Segmentation.

Number of Segments	Cut Discrepancy	Rand Index
2	0.002	0.05
4	0.03	0.17
8	0.36	0.37

Table 4.2: 64-Node Mesh Segmentation.

Number of Segments	Cut Discrepancy	Adjusted Rand Index
2	0.0003	0.07
4	0.012	0.11
8	0.32	0.15

Table 4.3: 128-Node Mesh Segmentation.

While all the algorithm runs above were carried out using a quantum simulator, the effect of noise of an actual quantum computer (from IBM) on the algorithm can be found in chapter 2. In addition, a study demonstrating the time taken by the quantum simulator versus the time taken to solve the same instance on real hardware is also shown in chapter 2. Even though this study is on another problem, this can help shed some light regarding how the runtime of the algorithm evolves with increasing size.

4.9 Conclusion

In this chapter we try to carry out mesh segmentation using a quantum algorithm. Using this algorithm we are able to perform mesh segmentation of size n using $\lceil \log_2 n \rceil$ qubits. This allows us to represent meshes of sizes up to 128 using only 7 qubits, something that would have taken 128 qubits with QAOA. This would mean that with 20 qubits we can reach a mesh size of 1 million which is around the industrial size of meshes. However, there are certain challenges that need to be faced in order to do this. The main challenge is that of the overhead due to Pauli decomposition. As shown in section 1.1.5, we need to calculate the coefficients of n^2 terms where n is the size of the mesh. For a mesh of size 1 million, we will need to calculate 10^{12} terms. There is already some interesting research to mitigate this issue. For example, much more efficient schemes to decompose the Hamiltonian and calculate the expectation value exist but only for d -sparse matrices [62].

We benchmark LogQ against the well-known KMeans algorithm. Although the quantum algorithm is able to find clusters, the quality of the results is far below that of KMeans as the number of segments increase. This is understandable for several reasons. One is the choice of objective function. The bi-partition version of modularity can do only two partitions and therefore requires multiple iterations to get multiple clusters. This could induce errors in the segmentation. A more appropriate and optimized objective function can improve performance significantly. As explained in section 4.6, we can generalize the definition of modularity to be able to divide the mesh into any number of segments. This increases the number of qubits required from $\lceil \log_2 n \rceil$ to $\lceil \log_2 2cn \rceil$. Alternatively, a totally different measure like within-cluster variance can be used.

Secondly, the performance also depends on the population size and number of the iterations of genetic algorithm in our algorithm and hence on computation time. The computation time increases linearly with the increase in either population size or number of iterations or both. As mentioned in the previous section, it is important to find the right balance between the performance and the computation time.

Nevertheless this serves as a first proof of concept on how the problem of mesh segmentation can be tackled using a quantum variational algorithm.

Conclusion

In this thesis we investigate quantum variational algorithms for combinatorial optimization problems. In addition, we also explore possible use-cases and tackled them using hybrid quantum algorithms. We identify that the contemporary quantum algorithms for optimization such as the Quantum Approximate Optimization algorithm (QAOA) scale linearly with problem size, limiting their applications to very small toy problems.

We present a novel way to encode optimization problems on a gate-based quantum computer such that the number of qubits required is reduced significantly - logarithmically, to be exact. We call this encoding the LogQ encoding. We first present an algorithm using this encoding for the MAXIMUM CUT problem. This is a good point to start because the MAXIMUM CUT problem has been of huge interest to the quantum computing community given its resemblance to the Ising Model in physics [90].

In order for the encoding to be applied to use-cases, however, it is important for it to be able to solve a wide range of problems. We therefore generalize the encoding using two approaches. The first one is using polynomial reductions. In 1972, Karp showed that NP-Hard problems are inter-convertible. We used this idea to convert other problems to the MAXIMUM CUT problems either directly or indirectly. The second approach is to use the Quadratic Unconstrained Binary Optimization (QUBO) matrix. We show that if a problem can be written down as a QUBO, then it can be encoded using the LogQ encoding. The QUBO in question must be Hermitian in order for it to be compatible with the quantum computer. Therefore, for problems which are difficult to convert to MAXIMUM CUT or it is very expensive (in terms of problem size) to do so, we can try to write it as a QUBO.

We benchmark LogQ against classical algorithms like Integer Linear Program and Gommans Williamson algorithm (a semi-definite programming algorithm). Note that the algorithms we present here are *heuristic* algorithm, like all variational algorithms. They do not, therefore, possess any proof of optimality and have different results on every run. We therefore run our algorithm several times and provide the average result to provide a fair representation of the performance. The experiments are carried out using both quantum simulators and IBM quantum computers up to graph sizes of 256. To our knowledge, these constitute the largest realistic combinatorial optimization problems ever run on a NISQ device, overcoming previous problem sizes by almost tenfold.

We then present a column generation algorithm for the Fleet Conversion use-case. Column generation deconstructs the problem into a multi-level problem where there is a coordinator or main problem and then there are sub-problems or worker problems. We use the commercial solver Gurobi to solve the coordinator problems and our algorithm using the LogQ encoding to solve the worker problems. This demonstrates that quantum algorithms can be used in tandem with the existing state of the art classical algorithms rather than being in competition with on another. Not only that, the LogQ encoding allowed us to reach

realistic problem sizes (up to 128 tours).

Finally we explore the problem of mesh segmentation. We represent the mesh as a graph and convert the problem into a problem of graph clustering. We choose modularity as our objective function and maximize it using the LogQ encoding. The results, however, were disappointing when compared to the classical KMeans clustering algorithm.

Perspectives

Using the LogQ encoding, we could reach an impressive problem size of 256 on quantum simulators as well as IBM quantum computers. There are, however, several factors due to which we could not go further. Firstly the LogQ encoding requires the efficient decomposition of the Hamiltonian matrix into Pauli strings. The method we apply is a brute force method and takes a significant amount of time. Hence, future work on the algorithm should incorporate efficient Pauli decomposition methods with the LogQ encoding. We have cited previous work where there exist some efficient ways to decompose the matrix but only when they are sparse.

Next, while we are able to reduce the number of qubits, the number of variables still remain the same as in a classical algorithm and therefore we are in the same search space as in a classical algorithm. A multi-solution approach to our encoding could be an interesting avenue to explore. By adding N -qubits to the model, we can add 2^N solutions that can be treated simultaneously and then at every iteration only the best ones can be kept, leading to a sort of quantum genetic algorithm.

Finally, The experiments on quantum computers are done only using the IBM quantum computers and hence it could be useful to see how this encoding can be implemented on other types of universal gate-based quantum computers.

Bibliography

- [1] O. Ezratty, *Understanding quantum technologies*. 2023.
- [2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011, ISBN: 9781107002173. [Online]. Available: <https://www.amazon.com/Quantum-Computation-Information-10th-Anniversary/dp/1107002176?SubscriptionId=AKIAI0BINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1107002176>.
- [3] J. Preskill, "Quantum computing 40 years later," *arXiv*, 2021. [Online]. Available: <https://www.amazon.science/publications/quantum-computing-40-years-later>.
- [4] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018, ISSN: 2521-327X. DOI: [10.22331/q-2018-08-06-79](https://doi.org/10.22331/q-2018-08-06-79). [Online]. Available: <https://doi.org/10.22331/q-2018-08-06-79>.
- [5] N. Gisin and R. Thew, "Quantum communication," *Nature photonics*, vol. 1, no. 3, pp. 165–171, 2007.
- [6] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dušek, N. Lütkenhaus, and M. Peev, "The security of practical quantum key distribution," *Reviews of modern physics*, vol. 81, no. 3, p. 1301, 2009.
- [7] M. Mehic, M. Niemiec, S. Rass, *et al.*, "Quantum key distribution: A networking perspective," *ACM Computing Surveys (CSUR)*, vol. 53, no. 5, pp. 1–41, 2020.
- [8] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum metrology," *Physical review letters*, vol. 96, no. 1, p. 010 401, 2006.
- [9] R. Orús, S. Mugel, and E. Lizaso, "Quantum computing for finance: Overview and prospects," *Reviews in Physics*, vol. 4, p. 100 028, 2019.
- [10] D. Herman, C. Googin, X. Liu, *et al.*, "A survey of quantum computing for finance," *arXiv preprint arXiv:2201.02773*, 2022.
- [11] B. A. Martin, P. Simon, and M. J. Rančić, "Simulating strongly interacting hubbard chains with the variational hamiltonian ansatz on a quantum computer," *Physical Review Research*, vol. 4, no. 2, p. 023 190, 2022.
- [12] M. Haidar, M. J. Rancic, Y. Maday, and J.-P. Piquemal, "Extension of the trotterized unitary coupled cluster to triple excitations," *The Journal of Physical Chemistry A*, vol. 127, no. 15, pp. 3543–3550, 2023.

- [13] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.
- [14] 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](https://doi.org/10.3390/a12020034). [Online]. Available: <https://www.mdpi.com/1999-4893/12/2/34>.
- [15] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973. DOI: [10.1147/rd.176.0525](https://doi.org/10.1147/rd.176.0525).
- [16] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, J. de Bakker and J. van Leeuwen, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 632–644, ISBN: 978-3-540-39346-7.
- [17] P. A. Benioff, "Quantum mechanical hamiltonian models of discrete processes that erase their own histories: Application to turing machines," *International Journal of Theoretical Physics*, vol. 21, pp. 177–201, 1982.
- [18] M. Cerezo, A. Arrasmith, R. Babbush, *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021. [Online]. Available: <https://www.nature.com/articles/s42254-021-00348-9>.
- [19] A. Peruzzo, J. McClean, P. Shadbolt, *et al.*, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, pp. 1–7, 2014. [Online]. Available: <https://www.nature.com/articles/ncomms5213>.
- [20] N. Moll, P. Barkoutsos, L. S. Bishop, *et al.*, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, no. 3, p. 030503, 2018. [Online]. Available: <https://iopscience.iop.org/article/10.1088/2058-9565/aab822>.
- [21] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, "Variational quantum algorithms for nonlinear problems," *Physical Review A*, vol. 101, no. 1, p. 010301, 2020. [Online]. Available: <https://journals.aps.org/prabstract/10.1103/PhysRevA.101.010301>.
- [22] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, "Quantum natural gradient," *Quantum*, vol. 4, p. 269, 2020. [Online]. Available: <https://quantum-journal.org/papers/q-2020-05-25-269/>.
- [23] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Physical Review X*, vol. 10, no. 2, p. 021067, 2020. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevX.10.021067>.
- [24] I. Q. Learning, *Cost functions*. [Online]. Available: <https://learning.quantum.ibm.com/course/variational-algorithm-design/cost-functions#the-estimator-primitive>.
- [25] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, "Perspectives of quantum annealing: Methods and implementations," *Reports on Progress in Physics*, vol. 83, no. 5, p. 054401, 2020.
- [26] R. M. Karp, "Reducibility among combinatorial problems," In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds) *Complexity of Computer Computations. The IBM Research Symposia Series*. Springer, Boston, MA., 1972. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4684-2001-2_9.

- [27] J. A. R.-V. et al., "Survey of polynomial transformations between np-complete problems," *Journal of Computational and Applied Mathematics*, vol. 235, pp. 4851–4865, 2011. [Online]. Available: <https://dl.acm.org/doi/abs/10.1016/j.cam.2011.02.018>.
- [28] J. C. Nash, "The (dantzig) simplex method for linear programming," *Computing in Science & Engineering*, vol. 2, no. 1, pp. 29–31, 2000.
- [29] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.
- [30] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," in *Operations Research Forum*, Springer, vol. 3, 2022, p. 20.
- [31] F. Glover, G. Kochenberger, and Y. Du, "Quantum bridge analytics i: A tutorial on formulating and using qubo models," *4OR, Springer*, vol. 17(4), pp. 335–371, 2019. [Online]. Available: <https://link.springer.com/article/10.1007/s10479-022-04634-2>.
- [32] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM (JACM)*, vol. 42, no. 6, pp. 1115–1145, 1995. [Online]. Available: <https://dl.acm.org/doi/10.1145/227683.227684>.
- [33] B. Gärtner and J. Matousek, *Approximation algorithms and semidefinite programming*. Springer Science & Business Media, 2012.
- [34] A. Pellow-Jarman, I. Sinayskiy, A. Pillay, and F. Petruccione, "A comparison of various classical optimizers for a variational quantum linear solver," *Quantum Information Processing*, vol. 20, no. 6, p. 202, 2021.
- [35] M. J. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*. Springer, 1994.
- [36] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.
- [37] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution," *arXiv preprint quant-ph/0001106*, 2000.
- [38] K. Blekos, D. Brand, A. Ceschini, et al., "A review on quantum approximate optimization algorithm and its variants," *Physics Reports*, vol. 1068, pp. 1–66, 2024, A review on Quantum Approximate Optimization Algorithm and its variants, ISSN: 0370-1573. DOI: <https://doi.org/10.1016/j.physrep.2024.03.002>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0370157324001078>.
- [39] C. Grange, M. Poss, and E. Bourreau, "An introduction to variational quantum algorithms for combinatorial optimization problems," *4OR*, vol. 21, no. 3, pp. 363–403, 2023.
- [40] C. J. Hillar and L.-H. Lim, "Most tensor problems are np-hard," *Journal of the ACM (JACM)*, vol. 60, no. 6, pp. 1–39, 2013. [Online]. Available: <https://dl.acm.org/doi/10.1145/2512329>.
- [41] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Combinatorial optimization—eureka, you shrink!* Springer, 2003, pp. 185–207. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-36478-1_17.

- [42] A. Sánchez-Arroyo, "Determining the total colouring number is np-hard," *Discrete Mathematics*, vol. 78, no. 3, pp. 315–319, 1989. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0012365X89901878>.
- [43] P. N. Klein and N. E. Young, "Approximation algorithms for np-hard optimization problems," in *Algorithms and theory of computation handbook: general concepts and techniques*, UC Riverside, 2010, pp. 34–34. [Online]. Available: <https://dl.acm.org/doi/10.5555/1882757.1882791>.
- [44] D. S. Hochba, "Approximation algorithms for np-hard problems," *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997. [Online]. Available: <https://dl.acm.org/doi/10.1145/261342.571216>.
- [45] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is np-hard," *Information Processing Letters*, vol. 42, no. 3, pp. 153–159, 1992. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/002001909290140Q>.
- [46] J. M. Hendrickx and A. Olshevsky, "Matrix p-norms are np-hard to approximate if $p \neq 1, 2, \infty$," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 5, pp. 2802–2812, 2010. [Online]. Available: <https://epubs.siam.org/doi/10.1137/09076773X>.
- [47] A. Callison and N. Chancellor, "Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond," *Phys. Rev. A*, vol. 106, p. 010 101, 1 Jul. 2022. DOI: [10.1103/PhysRevA.106.010101](https://doi.org/10.1103/PhysRevA.106.010101). [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevA.106.010101>.
- [48] K. M. Nakanishi, K. Fujii, and S. Todo, "Sequential minimal optimization for quantum-classical hybrid algorithms," *Physical Review Research*, vol. 2, no. 4, p. 043 158, 2020. [Online]. Available: <https://journals.aps.org/prresearch/abstract/10.1103/PhysRevResearch.2.043158>.
- [49] L. Bittel and M. Kliesch, "Training variational quantum algorithms is np-hard," *Physical Review Letters*, vol. 127, no. 12, p. 120 502, 2021. [Online]. Available: <https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.127.120502>.
- [50] N. Mariella and A. Simonetto, "A quantum algorithm for the sub-graph isomorphism problem," *arXiv preprint arXiv:2111.09732*, 2021. [Online]. Available: <https://arxiv.org/abs/2111.09732>.
- [51] P. K. Barkoutsos, G. Nannicini, A. Robert, I. Tavernelli, and S. Woerner, "Improving variational quantum optimization using cvar," *Quantum*, vol. 4, p. 256, 2020. [Online]. Available: <https://quantum-journal.org/papers/q-2020-04-20-256/>.
- [52] J. Tilly, H. Chen, S. Cao, *et al.*, "The variational quantum eigensolver: A review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0370157322003118>.
- [53] F. G. Fuchs, H. Ø. Kolden, N. H. Aase, and G. Sartor, "Efficient encoding of the weighted max k-cut on a quantum computer using qaoa," *SN Computer Science*, vol. 2, no. 2, pp. 1–14, 2021. [Online]. Available: <https://link.springer.com/article/10.1007/s42979-020-00437-z>.
- [54] J. Larkin, M. Jonsson, D. Justice, and G. G. Guerreschi, "Evaluation of qaoa based on the approximation ratio of individual samples," *Quantum Science and Technology*, 2022. [Online]. Available: <https://iopscience.iop.org/article/10.1088/2058-9565/ac6973>.

- [55] R. Herrman, L. Treffert, J. Ostrowski, P. C. Lotshaw, T. S. Humble, and G. Siopsis, "Globally optimizing qaoa circuit depth for constrained optimization problems," *Algorithms*, vol. 14, no. 10, p. 294, 2021. [Online]. Available: <https://www.mdpi.com/1999-4893/14/10/294>.
- [56] G. G. Guerreschi and A. Y. Matsuura, "Qaoa for max-cut requires hundreds of qubits for quantum speed-up," *Scientific reports*, vol. 9, no. 1, pp. 1–7, 2019. [Online]. Available: <https://www.nature.com/articles/s41598-019-43176-9>.
- [57] M. J. Rančić, "Noisy intermediate-scale quantum computing algorithm for solving an n-vertex maxcut problem with $\log(n)$ qubits," *Physical Review Research*, vol. 5, no. 1, p. L012021, 2023.
- [58] Y. Chatterjee, E. Bourreau, and M. J. Rančić, "Solving various np-hard problems using exponentially fewer qubits on a quantum computer," *Phys. Rev. A*, vol. 109, p. 052441, 5 2024. DOI: [10.1103/PhysRevA.109.052441](https://doi.org/10.1103/PhysRevA.109.052441).
- [59] Y. Chatterjee, M. J. Rančić, and E. Bourreau, "Logarithmic encoding of hamiltonians of np-hard problems on a quantum computer," *ROADEF*, 2023. [Online]. Available: <https://roadef2023.sciencesconf.org/434823>.
- [60] Y. Chatterjee, M. J. Rančić, and E. Bourreau, "Logarithmic encoding of hamiltonians of np-hard problems on a quantum computer," *EU/ME Metaheuristics Conference*, 2023. [Online]. Available: https://perso.isima.fr/~lacomme/GT2L/EUME_JE/papers/21_CHATTERJEE_soumission.pdf.
- [61] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM journal on matrix analysis and applications*, vol. 11(430), 1990. [Online]. Available: <https://epubs.siam.org/doi/10.1137/0611030>.
- [62] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. J. Coles, "Variational quantum linear solver," *Quantum*, vol. 7, p. 1188, 2023.
- [63] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders, "Efficient quantum algorithms for simulating sparse hamiltonians," *Communications in Mathematical Physics*, vol. 270, pp. 359–371, 2007.
- [64] M. P. Harrigan, K. J. Sung, M. Neeley, *et al.*, "Quantum approximate optimization of non-planar graph problems on a planar superconducting processor," *Nature Physics*, vol. 17, no. 3, pp. 332–336, 2021. [Online]. Available: <https://www.nature.com/articles/s41567-020-01105-y>.
- [65] D. Winderl, N. Franco, and J. M. Lorenz, "A comparative study on solving optimization problems with exponentially fewer qubits," *arXiv preprint arXiv:2210.11823*, 2022. [Online]. Available: <https://arxiv.org/pdf/2210.11823.pdf>.
- [66] J. C. Bezdek and R. J. Hathaway, "Some notes on alternating optimization," in *AFSS international conference on fuzzy systems*, Springer, 2002, pp. 288–300. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45631-7_39.
- [67] W. F. de la Vega and C. Kenyon-Mathieu, "Linear programming relaxations of max-cut," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07, New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 53–61, ISBN: 9780898716245. [Online]. Available: <https://dl.acm.org/doi/10.5555/1283383.1283390>.
- [68] P. Date, D. Arthur, and L. Pusey-Nazzaro, "Qubo formulations for training machine learning models," *Scientific reports*, vol. 11, no. 1, p. 10 029, 2021.

- [69] C. S. Calude, M. J. Dinneen, and R. Hua, "Qubo formulations for the graph isomorphism problem and related problems," *Theoretical Computer Science*, vol. 701, pp. 54–69, 2017.
- [70] C. Papalitsas, T. Andronikos, K. Giannakis, G. Theocharopoulou, and S. Fanarioti, "A qubo model for the traveling salesman problem with time windows," *Algorithms*, vol. 12, no. 11, p. 224, 2019.
- [71] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column generation*. Springer Science & Business Media, 2006, vol. 5.
- [72] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations research*, vol. 46, no. 3, pp. 316–329, 1998.
- [73] W. E. Wilhelm, "A technical review of column generation in integer programming," *Optimization and Engineering*, vol. 2, pp. 159–200, 2001.
- [74] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *Machine Learning*, vol. 46, pp. 225–254, 2002.
- [75] A. Mehrotra and M. A. Trick, "A column generation approach for graph coloring," *informs Journal on Computing*, vol. 8, no. 4, pp. 344–354, 1996.
- [76] Y. Chatterjee, Z. Allybokus, M. J. Rančić, and E. Bourreau, "A hybrid quantum-assisted column generation algorithm for the fleet conversion problem," *arXiv:2309.08267*, 2023.
- [77] E. Bourreau, Y. Chatterjee, Z. Allybokus, and M. J. Rančić, "Une génération de colonne assistée par machine quantique pour le problème de conversion de flotte de véhicules," *ROADEF*, 2024. [Online]. Available: <https://roadef2024.sciencesconf.org/510373>.
- [78] D. Bertsimas, C. Teo, and R. Vohra, "On dependent randomized rounding algorithms," *Operations Research Letters*, vol. 24, no. 3, pp. 105–114, 1999.
- [79] M. L. Balinski and A. W. Tucker, "Duality theory of linear programs: A constructive approach with applications," *SIAM Review*, vol. 11, no. 3, pp. 347–377, 1969, ISSN: 00361445. [Online]. Available: <http://www.jstor.org/stable/2028941> (visited on 09/14/2023).
- [80] D. Gale, H. W. Kuhn, and A. W. Tucker, "Linear programming and the theory of games," *Activity analysis of production and allocation*, vol. 13, pp. 317–335, 1951.
- [81] Y. Chatterjee, E. Bourreau, and H. Calandra, "D5.9: Benchmarking of qaoa-based algorithms for mesh segmentation, against k-means, normalized and randomized cuts and core extraction methods," 2024. [Online]. Available: https://www.neasqc.eu/wp-content/uploads/2024/05/Deliverable_D5_9_V0.2_VF.pdf.
- [82] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006. DOI: [10.1073/pnas.0601602103](https://doi.org/10.1073/pnas.0601602103). eprint: <https://www.pnas.org/doi/pdf/10.1073/pnas.0601602103>. [Online]. Available: <https://www.pnas.org/doi/abs/10.1073/pnas.0601602103>.
- [83] K. Bock and J. Stiller, "Optimizing triangular high-order surface meshes by energy-minimization," *Engineering with Computers*, vol. 34, no. 4, pp. 659–670, 2018.
- [84] B. Wang, G. Mei, and N. Xu, "Method for generating high-quality tetrahedral meshes of geological models by utilizing cgal," *MethodsX*, vol. 7, p. 101 061, 2020.

- [85] X. Jin and J. Han, "K-means clustering," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 563–564, ISBN: 978-0-387-30164-8. DOI: [10.1007/978-0-387-30164-8_425](https://doi.org/10.1007/978-0-387-30164-8_425). [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_425.
- [86] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3d mesh segmentation," *Acm transactions on graphics (tog)*, vol. 28, no. 3, pp. 1–12, 2009.
- [87] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971. DOI: [10.1080/01621459.1971.10482356](https://doi.org/10.1080/01621459.1971.10482356). eprint: <https://www.tandfonline.com/doi/pdf/10.1080/01621459.1971.10482356>. [Online]. Available: <https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356>.
- [88] Q. Huang and B. Dom, "Quantitative methods of evaluating image segmentation," in *Proceedings of the 1995 International Conference on Image Processing (Vol. 3)-Volume 3 - Volume 3*, ser. ICIP '95, USA: IEEE Computer Society, 1995, p. 3053, ISBN: 0818673109.
- [89] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: [10.1109/TIT.1982.1056489](https://doi.org/10.1109/TIT.1982.1056489).
- [90] B. A. Cipra, "An introduction to the ising model," *The American Mathematical Monthly*, vol. 94, no. 10, pp. 937–959, 1987.