

HYBRID QUANTUM OPTIMIZATION ALGORITHMS AND APPLICATIONS



Artemis Iliopoulou

Supervising Professor: Dimitrios Aggelakis

Examination Committee: Professor Vasilis Samoladas

Examination Committee: Professor Michail Zervakis

TECHNICAL UNIVERSITY OF CRETE

School of Electrical and Computer Engineering

Chania 2024

Abstract

This thesis explores the application of quantum computing in solving optimization and, more specifically, scheduling problems. The first chapter provides an introduction to quantum computing. It covers introductory definitions, including single and multiple qubit states, quantum gates, and entanglement. It also analyzes the construction and importance of one of the fundamental quantum algorithms, the Deutsch-Josza algorithm. The second chapter focuses on the basic approaches to quantum optimization, such as Quadratic Unconstrained Binary Optimization (QUBO), the Ising model, and the notions of quantum annealing and adiabatic quantum computing. It also analyzes the Quantum Approximate Optimization Algorithm (QAOA) and demonstrates the MaxCut problem and how it can be solved using the QAOA. The third chapter delves into scheduling problems, precisely the Nurse Scheduling Problem (NSP). It explores two different QUBO models for JSSP in the Appendix and presents the results of the simulations conducted on various instances of NSP problems. The quantum algorithm ran on the quantum cloud service in D-Wave to obtain the results. Then, the performance for different problem sizes and topologies of the hardware is analyzed. Overall, this thesis presents a comprehensive overview of the fundamental concepts and techniques used in hybrid quantum-classical optimization and demonstrates its potential for solving optimization and scheduling problems.

"Quantum mechanics describes nature as absurd from the point of view of common sense. And yet it fully agrees with experiment. So I hope you can accept nature as She is - absurd."

RICHARD P. FEYNMAN

Acknowledgements

A heartfelt thank you to my thesis supervisor, Professor Dimitris G. Angelakis, for his guidance throughout this journey. His wealth of knowledge and expertise in the field has been inspiring. I would also like to acknowledge the chance I had to interact with our collaborators at the Centre for Quantum Technologies at the National University of Singapore. Especially the useful weekly joint group meetings online over the last two years, the research seminars, and journal clubs, as well as Ben Tan, a former PhD student of the group whose inspiring research papers with Prof Angelakis in quantum optimization were useful for my work. I am also grateful to my family for their constant support and understanding. Thank you for always being there for me. Additionally, I am deeply thankful to my friends for their endless encouragement throughout my academic years. Without their support, this achievement would not be possible.

Contents

Abstract	1
Acknowledgements	3
List of Figures	6
List of Tables	11
Introduction	12
1 Introduction to Quantum Computing	15
1.1 Qubit	15
1.2 Single Qubit States	16
1.2.1 Mathematical Notation for Qubit	16
1.2.2 Properties of Notation	18
1.2.3 Visual Notation for Qubit	19
1.2.4 The Bloch Sphere	20
1.3 Multiple Qubit States	21
1.4 Single Qubit Gates	23
1.5 Two Qubit Gates	28
1.6 Quantum Entanglement	32
1.7 Deutsch- Jozsa algorithm	35
1.7.1 The Algorithm	37
2 Optimization: from classical to quantum	40
2.1 Combinatorial Optimization	41

2.2	Quadratic Unconstrained Binary Optimization (QUBO)	42
2.2.1	From QUBO to Hamiltonian	46
2.3	Ising Model	48
2.3.1	Ising Model and QUBO Model	49
2.3.2	Solving MaxCut Problem - Hamiltonian	52
2.4	Quantum Adiabaticity	56
2.4.1	Quantum Annealing	57
2.5	Variational Hybrid Quantum-Classical Algorithms	59
2.6	Quantum Approximate Optimization Algorithm (QAOA)	61
2.6.1	Cost Layer	64
2.6.2	Mixer Layer	67
2.6.3	Solving MaxCut Problem with Quantum Approximate Optimiza- tion Algorithm	70
3	Quantum Optimization for Scheduling Problems	76
3.1	Scheduling Problems	76
3.2	Nurse Scheduling Problem (NSP)	78
3.2.1	Problem Formulation	79
3.2.2	Objective Function Construction	82
3.2.3	Quantum Annealing at Play	95
3.2.4	Solution Checking	100
4	Conclusion and Future Work	104
A	Job Shop Scheduling Problem (JSSP)	106
A.1	Job Shop Scheduling Problem (JSSP)	106
A.1.1	Problem Definition	106
A.1.2	Disjunctive QUBO Model	112
A.1.3	Time-indexed QUBO Model	115
	Bibliography	119

List of Figures

1	From abacus to quantum computing. Source left, right	12
2	A Brief History of Quantum Computing. Source	13
3	History of quantum algorithms. Source	14
1.1	Object in superposition $ +\rangle$	17
1.2	Bloch Sphere	20
1.3	Bloch vector of some basic states	21
1.4	Identity Gate	24
1.5	X Gate	24
1.6	Y Gate	25
1.7	Z Gate	25
1.8	Hadamard Gate	26
1.9	CNOT Gate.	29
1.10	SWAP Gate symbol	31
1.11	CCNOT Gate	31
1.12	Fredkin Gate	31
1.13	Quantum circuit to create Bell state	33
1.14	Two cases: Constant or Balanced function.	36
1.15	Black Box	36
1.16	Deutsch- Jozsa algorithm quantum circuit	37
2.1	Heat-map visualization of the QUBO matrix	45
2.2	Bar chart of the solution vector, indices 1.0, 2.0, 3.0 represent our 3-bit solution	45

2.3	A lattice of sites where blue sites are in state $\sigma=+1$ and red sites are in state $\sigma=-1$	48
2.4	A lattice from a different angle.	48
2.5	Global and local minimum	49
2.6	QUBO graph of our problem	50
2.7	Ising graph of our problem	52
2.8	The dark blue and the light blue nodes are the two sets. In the output, with red color, are the edges that are cut.	52
2.9	Quantum adiabatic evolution in quantum annealing. If the evolution time (τ) is long enough, the Hamiltonian that controls the system will change slowly, which will allow the system to transition to a new ground state using the quantum tunneling effect.	57
2.10	Quantum Annealing functions A(s) -Initial Hamiltonian Energy- and B(s) -Final Hamiltonian Energy with linear anneal.	58
2.11	A hybrid quantum-classical computational approach with the parameterized quantum circuit running on the quantum computer and then outsourcing the parameter optimization to a classical optimizer.	59
2.12	Diagram of a VQA	60
2.13	A diagram of how the variational QAOA works	62
2.14	First Look of the QAOA Circuit Diagram	63
2.15	Implementation of cost Hamiltonian; shows how 2 CNOT and 1 \mathbf{R}_Z gate is equivalent with an \mathbf{R}_{ZZ} gate	67
2.16	Quantum circuit for the unitary operation	67
2.17	Implementation of mixer Hamiltonian	68
2.18	Implementation of gate	69
2.19	A 5-nodes fully connected graph	70
2.20	Mixer layer- apply on each qubit the \mathbf{R}_x gate parameterized by 2β	71
2.21	Cost layer- apply 2 CNOT gates and an \mathbf{R}_z gate (decomposition of \mathbf{R}_{zz} gate)	71

2.22	QAOA circuit - Between each barrier (grey dotted lines), there are the mixer operators $U(H_e, \gamma)$ for each edge	71
2.23	After 33 evaluations, we get $M_{p=1}(\gamma, \beta) = 5.91$	72
2.24	Distribution of states for the fully connected graph with optimal parameters	72
2.25	17-nodes fully connected graph	73
2.26	Plot of the cost functions of the QAOA over iteration steps for $p = 1$. . .	73
2.27	A heatmap for different values of parameters γ, β . Note how the better solutions are represented by darker colors (lower cost values) and the worst solutions are represented by brighter colors (higher cost values). .	74
2.28	Iteration/Cost function pairs	74
2.29	Plot of the cost functions of the QAOA over iteration steps for $p = 2$. . .	75
3.1	The classification of scheduling problems according to Nagar, Haddock & Heragu, 1995	77
3.2	Types of Scheduling Problems. Source	78
3.3	Schedule of the example	83
3.4	A representation of matrix J when solving the nurse scheduling problem for $N=3$ and $D=2$, with penalty $a=1.0$	84
3.5	Left: the qubit orientation for 72 qubits (36 horizontal and 36 vertical). Center: Vertical green qubits couples internally with 4 horizontal black qubits. Right: Horizontal green qubits couples externally with 2 horizontal blue qubits in adjacent unit cells. Source	89
3.6	Coupled qubits roadway-style in a 144 nodes-sized Pegasus processor. Qubits: dots, couplers: lines. Curved blue lines represent "internal" couplers, while long red lines represent "external" couplers and short red lines represent "odd" couplers.	89
3.7	Zephyr topology. Representative qubit (black dot) connected to orthogonal qubits by 16 internal couplers (green lines). External couplers (blue lines), odd couplers (red lines)	90
3.8	Left: The graph of QUBO problem with 3 variables, Right: The embedded problem in QPU that uses 4 qubits	91

3.9	A schedule that satisfies all the constraints of this NSP instance.	92
3.10	Schedules that do not satisfy all the constraints.	92
3.11	For the case of $N=2$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).	96
3.12	For the case of $N=3$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).	96
3.13	For the case of $N=4$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).	97
3.14	For the case of $N=2$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state by Reverse Annealing using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).	98
3.15	For the case of $N=3$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state by Reverse Annealing using D-Wave's QPU with two different topologies: Pegasus (left) and Zephyr (right).	99
3.16	For the case of $N=4$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state by Reverse Annealing using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).	100
3.17	The Hamming distance between consecutive Forward Annealing solutions concerning a number of days for $N= 2$ nurses (left), $N=3$ nurses (middle), and $N =4$ nurses (right), with Pegasus.	101
3.18	The Hamming distance between consecutive Reversed Annealing solutions with respect to some scheduled days for $N= 2$ nurses (left), $N=3$ nurses (middle), and $N =4$ nurses (right), with Pegasus.	101
3.19	The fraction of solutions that are in the ground state for the NSP with Forward Annealing by using the Pegasus topology (blue) and the Zephyr topology(red).	102

3.20 The fraction of solutions that are in the ground state for the NSP with Reverse Annealing by using the Pegasus topology (blue) and the Zephyr topology(red).	103
A.1 Problem solution in Gantt Chart.	109
A.2 Graph representation	111

List of Tables

1.1	Possible states for qubits	21
1.2	Basic non-unitary Matrices	26
1.3	Relation of Pauli gates with basic matrices	26
1.4	Action of CNOT gate	29
1.5	Truth table of XOR gate	29
1.6	Action of CZ gate	30
1.7	Action of SWAP gate	31
2.1	Table of known constraint/penalty pairs.	46
3.1	Possible states for 2 Nurses	93
3.2	Possible Assignments for a problem with 2 Nurses and 2 Days	93
A.1	Route of machines	108
A.2	Completion time of jobs	110
A.3	Completion time of jobs	110

Introduction

Human evolution over many years was driven by persistent curiosity, which fuels people's long journeys with innovation. From ancient times, people have continuously developed and refined essential tools to improve their daily lives. One such example is the abacus, a foundational instrument for mathematical calculations that has undergone numerous advancements over the years. All those innovations through the years laid the groundwork for the remarkable advancements in computing and technology witnessed throughout the 20th century. Alan Turing was a visionary who successfully managed to mark the classical computing era, while Max Planck and Albert Einstein achieved significant milestones in quantum theory. It was only in 1913 that Niels Bohr introduced the idea of quantum jumps and the Bohr model of the atom. After that, it followed the revolutionary Schrödinger equation formulated in 1926 by Erwin Schrödinger, as well as the principle of uncertainty established by Werner Heisenberg the following year, both of which were pivotal developments in the field. Nevertheless, genuine innovations in quantum theory have not yet been made.

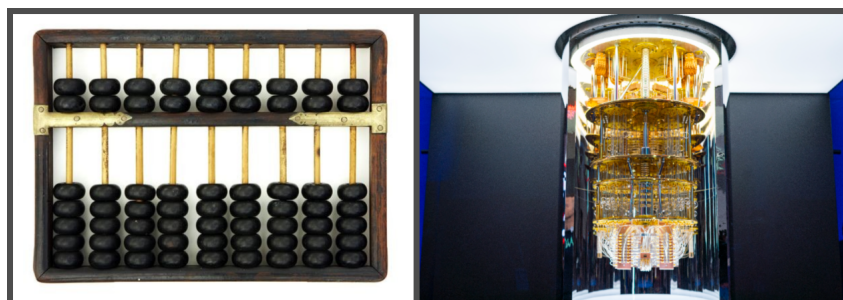


Figure 1: From abacus to quantum computing. Source left, right

The beginning of quantum computing didn't occur until the latter half of the 20th

century when the PhysComp conference in 1980 took place. There, scientists got together and exchanged ideas, marking an era that shaped the field and the idea of quantum computation as we know it today. The first model for a quantum computer was proposed in 1981 by Richard Feynman. In the ensuing years, the groundwork for quantum computing was laid, such as the development of quantum logic gates. However, real innovations in quantum theory and the beginning of quantum computing started in the latter half of the 20th century.

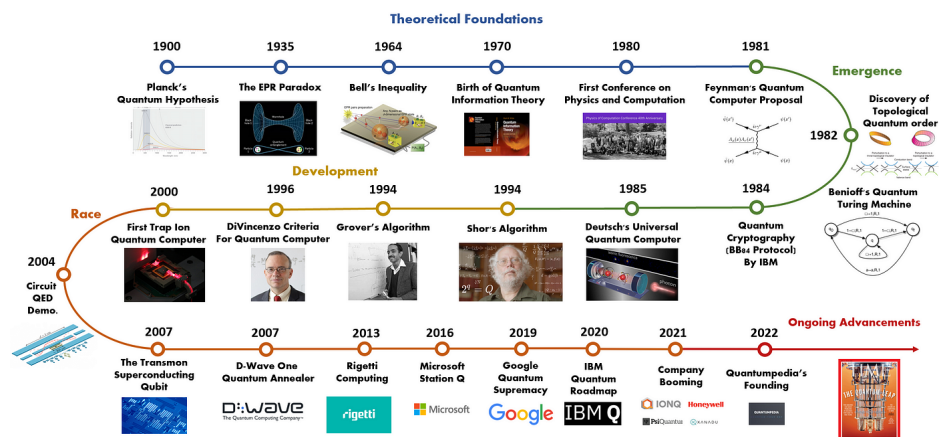


Figure 2: A Brief History of Quantum Computing. Source

A game-changing work in the quantum computing world was in 1992 when David Deutsch and Richard Josza proposed the Deutsch-Josza algorithm. This algorithm was one of the first to demonstrate the power of quantum computation since it surpasses the classical algorithm. After that, Peter Shor and Lov Grover presented some groundbreaking work, too. Shor's algorithm, proposed in 1994, could potentially solve the integer factorization problem used in cryptography, such as RSA and ECC encryption, efficiently in polynomial time on quantum computers, whereas classical computers need exponential time. Two years later, in 1996, Lov Grover designed Grover's algorithm, which offers quadratic speedup over the classical algorithms for searching unstructured databases. Those two algorithms had a significant impact because they were the first to show the potential of quantum computing, which inspired others to do further research in the field.

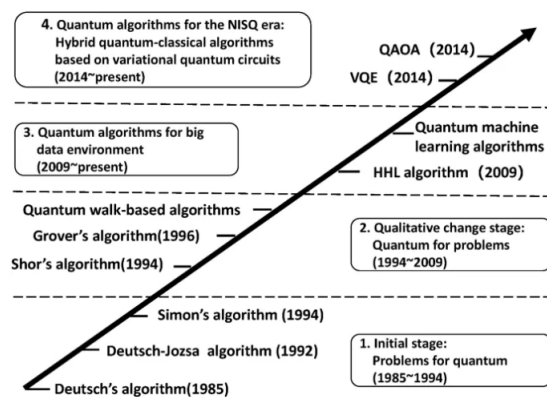


Figure 3: History of quantum algorithms. Source

Nowadays, we see a rapid evolution in quantum computing, and billions of dollars are being invested in the field. Companies, governments, and institutions worldwide are trying to achieve the next breakthrough. Today, several companies, such as IBM, Google, Amazon, Rigetti, and IonQ, own quantum computers with around 100 qubits. IBM recently announced that they released a quantum chip with 1,121 qubits. However, we should keep in mind that quantum computers are still "noisy" and incredibly fragile. Even the task of ensuring the stability of the qubits is very challenging since, until now, cryogenic temperatures are required. Overall, quantum computing is still at an early stage, with some comparing the quantum computers with the classical ones in the 50s because they are yet large, expensive, and limited in functionality, but the research shows promising potential. Currently, scientists worldwide are working on building a working fault-tolerant quantum computer.

Chapter 1

Introduction to Quantum Computing

This chapter begins with the definition of qubit, the core of quantum computing. We will discuss the single and multiple qubit states as well as the mathematical notation and properties. We will mention the Bloch sphere and how to represent a qubit visually. We will introduce the operations that can change the state of qubits, the so-called gates. Then, we will discuss an essential phenomenon of the quantum world, the entanglement, and its role in quantum computing. Finally, we will present our first algorithm, the Deutsch-Jozsa algorithm, and we will understand its importance. Through this chapter, we aim to understand the fundamental principles of quantum computing, which will be helpful later on.

1.1 Qubit

The fundamental unit of information in quantum computing is known as a qubit, which is short for "quantum bit". The term qubit was invented by Benjamin Schumacher along with William Wootters, two theoretical physicists, in the late 1990s. It's a quantum analog of the classical bit, but with unique properties derived from the principles of quantum mechanics. In terms of storing information, a qubit plays a similar role as a bit, but its behavior is based on quantum properties. While classically, we traditionally

have either 0 or 1; a qubit is not limited to just the $|0\rangle$ and $|1\rangle$ states but instead can exist in a combination between $|0\rangle$ and $|1\rangle$ states.

1.2 Single Qubit States

1.2.1 Mathematical Notation for Qubit

The most widely used formulation to describe quantum states is the Dirac Notation or Bra-ket Notation -*(bra|ket)*- which was introduced in 1939 by Paul Dirac. A quantum state is represented by the ket, which is of the form $|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$, while the same vector as bra is of the form $\langle\psi| = [a^*b^*]$. So bra is the Hermitian conjugate of ket, and the inner product is

$$\langle\phi | \psi\rangle = c \text{ (a number)}$$

If the inner product is denoted as the expression $c = \langle\phi | \psi\rangle$ then the complex conjugate c^* of this expression is $c^* = \langle\phi | \psi\rangle^* = \langle\phi | \psi\rangle$. Kets and bras exist in a Hilbert space, usually symbolized by \mathcal{H} . This space serves as a generalization of the three-dimensional linear vector space found in Euclidean geometry, extending it to a complex-valued space that can accommodate a potentially infinite number of dimensions.

The simplest Hilbert space we can examine is the two-level quantum systems, with the two vectors $|0\rangle$ and $|1\rangle$ forming an orthonormal basis for the vector space. These special states are known as computational basis states.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The two basic states in the form of a bra are

$$\langle 0| = \begin{bmatrix} 1 & 0 \end{bmatrix} \text{ and } \langle 1| = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

For the two basic states we mentioned that are an orthonormal base, which means their magnitudes must be equal to one. So, these vectors must also be orthogonal to each other, which means that their inner product must be equal to zero. We show that

$$\langle 0|0\rangle = \langle 1|1\rangle = 1$$

$$\langle 0|1\rangle = \langle 1|0\rangle = 0$$

It is possible to form linear combinations of states where an object is in two distinguishable states, called superposition:

$$|\psi\rangle = \alpha |0\rangle + b |1\rangle$$

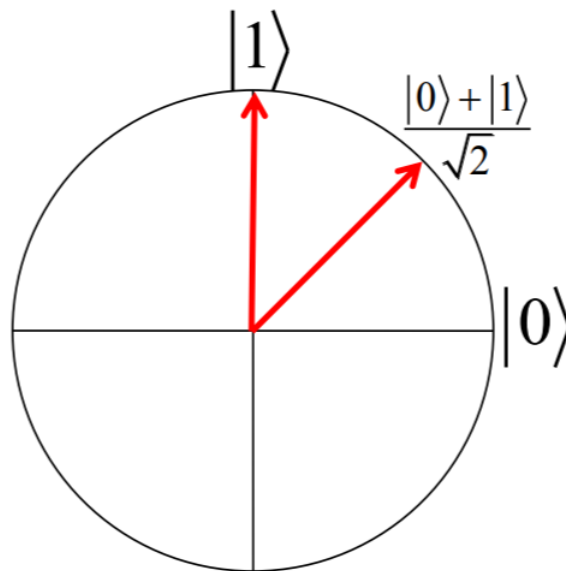


Figure 1.1: Object in superposition $|+\rangle$.

In quantum computing, we cannot determine whether a bit is in state 0 or 1 like in classical computers, since the quantum state is the complex values of α and b . When a qubit is being measured, we observe either the result 0, with probability $|a|^2$, or the result 1, with probability $|b|^2$. This phenomenon happens because when we measure the state of a qubit, the superposition is lost, and its state collapses to a state of 0 or 1. Although nobody knows why this type of collapse occurs, this unique behavior is a fundamental principle of quantum mechanics. Note that the probabilities must always sum to one, $|a|^2 + |b|^2 = 1$.

1.2.2 Properties of Notation

The inner product is linear

$$\langle \phi | (a_1 | \psi_1 \rangle + a_2 | \psi_2 \rangle) = a_1 \langle \phi | \psi_1 \rangle + a_2 \langle \phi | \psi_2 \rangle.$$

Suppose two kets $|\psi\rangle, |\chi\rangle$

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} = a |0\rangle + b |1\rangle, \quad |\chi\rangle = \begin{bmatrix} c \\ d \end{bmatrix} = c |0\rangle + d |1\rangle$$

Their inner product is of the form

$$\langle |\chi\rangle, |\psi\rangle \rangle = \langle \chi | \psi \rangle = \langle \chi | \cdot | \psi \rangle = [c^* \ d^*] \begin{bmatrix} a \\ b \end{bmatrix} = c^* a + d^* b$$

$$\langle |\psi\rangle, |\chi\rangle \rangle = \langle \psi | \chi \rangle = \langle \psi | \cdot | \chi \rangle = [a^* \ b^*] \begin{bmatrix} c \\ d \end{bmatrix} = a^* c + b^* d$$

Note that $\langle \chi | \psi \rangle \neq \langle \psi | \chi \rangle$.

It is noteworthy that the inner product of a state with itself represents the probability of finding the system in that state upon measurement. This probability is given by the square of the magnitude of the state vector.

$$\langle \psi | \psi \rangle = \langle \psi | \cdot | \psi \rangle = [a^* \ b^*] \begin{bmatrix} a \\ b \end{bmatrix} = a^* a + b^* b = |\psi|^2$$

Their outer product is of the form

$$|\psi\rangle \langle \chi| = \begin{bmatrix} a \\ b \end{bmatrix} [c^* \ d^*] = \begin{bmatrix} ac^* & ad^* \\ bc^* & bd^* \end{bmatrix}$$

$$|\chi\rangle \langle \psi| = \begin{bmatrix} c \\ d \end{bmatrix} [a^* \ b^*] = \begin{bmatrix} ca^* & cb^* \\ da^* & db^* \end{bmatrix}$$

Note that $|\psi\rangle \langle \chi| \neq |\chi\rangle \langle \psi|$ but $(|\psi\rangle \langle \chi|)^\dagger = |\chi\rangle \langle \psi|$

Quantum computing involves the use of operations, which can either be physical processes or mathematical transformations that are applied to a quantum system. These operations are crucial in quantum computing since they enable the manipulation of a quantum system's properties and states.

Operations are usually denoted by a hat \hat{A} .

$$\hat{A}(c_1|\psi_1\rangle + c_2|\psi_2\rangle) = c_1\hat{A}|\psi_1\rangle + c_2\hat{A}|\psi_2\rangle.$$

The matrix element of an operator is

$$\langle\phi|\hat{A}|\psi\rangle = \langle\phi|(\hat{A}|\psi\rangle) = (\langle\phi|\hat{A})|\psi\rangle = c \text{ (a number)}.$$

The expectation value of an operator for a system in the state $|\psi\rangle$ is

$$\langle\hat{A}\rangle = \langle a \rangle = \langle\psi|\hat{A}|\psi\rangle.$$

The complex conjugate of the matrix element is

$$\langle\phi|\hat{A}|\psi\rangle^* = \langle\psi|\hat{A}^\dagger|\phi\rangle = c^*$$

where \hat{A}^\dagger is the Hermitian conjugate of \hat{A} .

1.2.3 Visual Notation for Qubit

Qubits can also be conceptualized as quantum analogs of spins, which are fundamental properties of subatomic particles like electrons or photons. The nature and the behavior of each of these particles, known as qubits, form the basis of quantum computing. When used as a qubit, a particle is placed in a controlled environment (e.g., floating in a magnetic field) that protects and isolates it from the outer environment.

In general, we can encode qubit states into the direction of the electron's spin. The two fundamental states are the up-spin ($|\uparrow\rangle$) and the down-spin ($|\downarrow\rangle$). The first is the state that corresponds to the electron's spin aligned in the up direction, while the following corresponds to the electron's spin aligned in the down direction.

$$|0\rangle = |+\rangle = |\uparrow\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = |-z\rangle = |\downarrow\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

1.2.4 The Bloch Sphere

The Bloch sphere, named after the physicist Felix Bloch, is a graphical representation that provides the visualization of a single qubit's state by rotating an “arrow”-vector in a three-parameter (x, y, z) real space. Note that by choosing a three-dimensional sphere, we can represent all the possible state-vectors. The “arrow's” point on the sphere can be represented as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\varphi} \sin\left(\frac{\theta}{2}\right) |1\rangle$$

The parameters are ranged with $0 \leq \theta \leq \pi$ and $0 \leq \varphi \leq 2\pi$. Note that

$$\left|\cos\left(\frac{\theta}{2}\right)\right|^2 + \left|e^{i\varphi} \sin\left(\frac{\theta}{2}\right)\right|^2 = 1$$

which means that it is a valid state since the probabilities sum up to one.

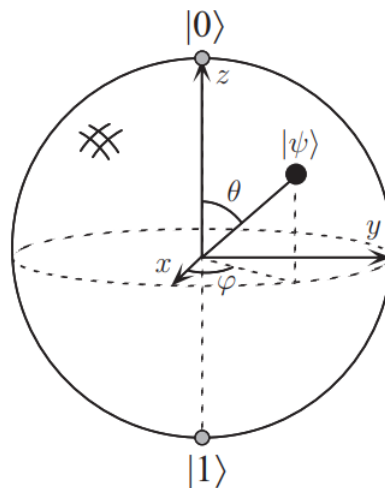


Figure 1.2: Bloch Sphere

From figure 1.2, it is easy to understand that the parameter θ is an angle from z -axis to $|\psi\rangle$ and so it is always positive. Furthermore, the angle φ is being counted from x -axis to the y -axis.

The figure 1.3 illustrates the Bloch vector of a few fundamental states, providing a visual representation of them.

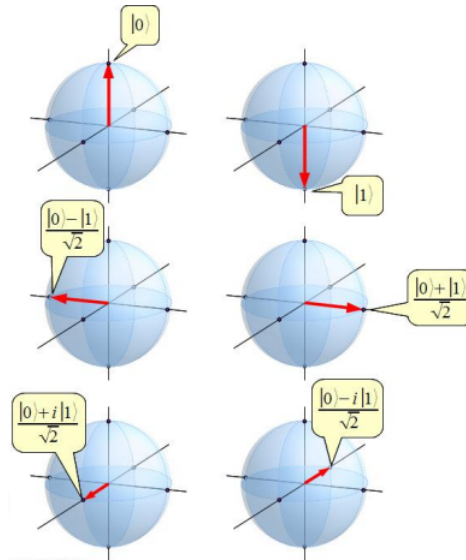


Figure 1.3: Bloch vector of some basic states

1.3 Multiple Qubit States

Just like two classical bits have four possible states; $\{00, 01, 10, 11\}$, in quantum, the possible states for a two-qubit system are also four; $\{ |00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. Similarly, for a three-qubit system, the available state space is $\{ |000\rangle, |001\rangle, |010\rangle, \dots, |110\rangle, |111\rangle\}$. It is easy to notice that the number of possible states grows exponentially as the number of qubits increases, with the possible states being 2^n .

Number of qubits	Number of possible states
2	$(2^2 = 4)$
5	$(2^5 = 32)$
10	$(2^{10} = 1024)$
50	$(2^{50} = 1126 \times 10^{15})$ (approximate)
100	$(2^{100} = 1268 \times 10^{30})$ (approximate)

Table 1.1: Possible states for qubits

So we describe a quantum state of n qubits: $\sum_{x \in \{0,1\}^n} a_x |x\rangle$

We should mention that when we write $|vw\rangle$ or $|v, w\rangle$, $|v\rangle |w\rangle$, for example $|00\rangle$, we

use the abbreviated notation for the tensor product $|v\rangle \otimes |w\rangle$.

Definition 1.3.1. *Suppose V and W are Hilbert spaces of dimension m and n respectively. Then $V \otimes W$ is an mn dimensional vector space. The elements of $V \otimes W$ are linear combinations of ‘tensor products’ $|v\rangle \otimes |w\rangle$ of elements $|v\rangle$ of V and $|w\rangle$ of W . In particular, if $|i\rangle$ and $|j\rangle$ are orthonormal bases for the spaces V and W then $|i\rangle \otimes |j\rangle$ is a basis for $V \otimes W$.*

The tensor product of the vectors (a, b) and (c, d) is the vector

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

The tensor product of two 2×2 matrices is the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a & \begin{bmatrix} e & f \\ g & h \end{bmatrix} \\ c & \begin{bmatrix} e & f \\ g & h \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{bmatrix}$$

Now we can easily define the basis vectors

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

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

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

In general, the superposition property allows a quantum computer to exist in multiple states simultaneously. For example, a quantum state of two qubits can exist in the superposition of all four states, and so we involve a complex coefficient with each computational basis state

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle$$

The probability of the result x after the measurement is denoted as $|a_x|^2$ with $x = \{00, 01, 10, 11\}$ and we have $\sum_{x \in \{0,1\}^2} |a_x|^2 = 1$ because the probabilities sum to one.

1.4 Single Qubit Gates

Single qubit gates play a crucial role in quantum computing as they are the fundamental quantum operations that transform the state of a single qubit. It's important to note that all quantum gates are unitary matrices, and through their multiplication, the direction of the vector changes in the Hilbert space. This means that the effect of these gates on the state of the qubit can be thought of as rotating the qubit's state vector around the Bloch sphere.

The most frequently used single qubit gates in quantum computing are the four Pauli gates.

The first gate is known as the Identity Gate (I, σ_0). This operation leaves the qubit state unchanged. The Identity Gate is represented by

$$I = |0\rangle\langle 0| + |1\rangle\langle 1|$$

The matrix representation of Identity Gate is $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and the circuit symbol is

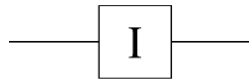


Figure 1.4: Identity Gate

It is obvious that the action of the gate is $I |0\rangle = |0\rangle$ and $I |1\rangle = |1\rangle$.

The second is the Pauli-X Gate. It is also known as the NOT gate and is represented by symbols such as $(X, NOT, \sigma_x, \sigma_1)$. This gate behaves similarly to the classical NOT gate, but instead of flipping a classical bit, it flips the quantum state of a qubit. Specifically, it converts the quantum state of the qubit from $|0\rangle$ to $|1\rangle$ and vice versa. Mathematically, it is represented by

$$X = |0\rangle\langle 1| + |1\rangle\langle 0|$$

The matrix representation of the Pauli-X gate is $\sigma_x = X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and the circuit symbol is shown in the figure 1.5

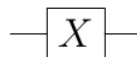


Figure 1.5: X Gate

The action of X Gate is shown $X |0\rangle = |1\rangle$ and $X |1\rangle = |0\rangle$.

The third gate used in quantum computing is known as the Pauli-Y Gate (Y, σ_y) . This gate is like the x gate as it flips a bit, but it also adds a phase.

$$Y = i|1\rangle\langle 0| - i|0\rangle\langle 1|$$

Its matrix representation is $\sigma_y = Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ and the circuit symbol is shown in the figure 1.6

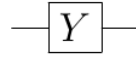


Figure 1.6: Y Gate

When applied to the quantum states $|0\rangle$ and $|1\rangle$, the action of the Y gate is as follows: $Y |0\rangle = i|1\rangle$ and $Y |1\rangle = -i |0\rangle$.

The fourth and the last of the Pauli Gates is the Z gate (Z, σ_z), which adds a phase flip to the qubit. Represented by the circuit symbol displayed in Figure 1.7, its form is

$$Z = |0\rangle\langle 0| - |1\rangle\langle 1|$$

The Z Gate can be expressed as a matrix with the values: $\sigma_z = Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

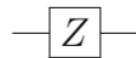


Figure 1.7: Z Gate

The function of the Z Gate can be understood as follows: it flips the phase of a qubit. For example, $Z |0\rangle = |0\rangle$ and $Z |1\rangle = - |1\rangle$.

In table 1.2, we mention a list of the basic non-unitary matrices that are not suitable for use as quantum gates as they are not unitary. However, they can be utilized in combinations to achieve desired results.

Moving on to Table 1.3, we observe the relationship between the Pauli gates and the aforementioned matrices.

One of the most well-known gates in quantum computing is the Hadamard Gate often denoted as the 'H-gate'. This gate operates on a single qubit and is fundamental because it can create superposition states, which are a critical part of quantum computing. More specifically, the Hadamard gate maps the input qubit state to a superposition of the $|0\rangle$ and $|1\rangle$ basis states. This means that when the qubit is measured in the computational

$$\begin{aligned}
|0\rangle\langle 0| &= \begin{bmatrix} 1 & \\ & 0 \end{bmatrix} [1 \ 0] = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} &= |0\rangle\langle 0| = \frac{1}{2}(I + \sigma_z) \\
|0\rangle\langle 1| &= \begin{bmatrix} 1 & \\ & 0 \end{bmatrix} [0 \ 1] = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} &= |0\rangle\langle 1| = \frac{1}{2}(\sigma_x + i\sigma_y) \\
|1\rangle\langle 0| &= \begin{bmatrix} 0 & \\ & 1 \end{bmatrix} [1 \ 0] = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} &= |1\rangle\langle 0| = \frac{1}{2}(\sigma_x - i\sigma_y) \\
|1\rangle\langle 1| &= \begin{bmatrix} 0 & \\ & 1 \end{bmatrix} [0 \ 1] = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} &= |1\rangle\langle 1| = \frac{1}{2}(I - \sigma_z)
\end{aligned}$$

Table 1.2: Basic non-unitary Matrices

Table 1.3: Relation of Pauli gates with basic matrices

basis, there is an equal chance of 50% to observe the qubit in the $|0\rangle$ state and 50% probability of observing the qubit in the $|1\rangle$ state. The Hadamard gate, if applied to a qubit, can be represented as:

$$H |0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \text{ and } H |1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Its matrix representation can be expressed as $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(\sigma_x + \sigma_z)$

It should be noted that the Hadamard gate can transform the state of a qubit, but this transformation is reversible. By applying the Hadamard gate again, the original state of the qubit can be restored. This characteristic makes the Hadamard gate an essential building block in quantum computing which is used in many quantum algorithms and protocols. The figure 1.8 illustrates the circuit symbol.

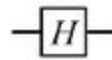


Figure 1.8: Hadamard Gate

There exist a lot more crucial single qubit gates, such as the phase gate and the rotation gates. The phase gate introduces a phase shift to the state by rotating the qubit around the z-axis by a fixed angle ϕ . The rotation gates, $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$, are categorized by the axis and the angle of the rotation and perform a continuous transformation on the qubit state.

The phase shift gate, denoted as $P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}$, acts on qubits as follows:

$$P(\varphi) |0\rangle = |0\rangle \text{ and } P(\varphi) |1\rangle = e^{i\varphi} |1\rangle.$$

It is important to note that the probability of measuring either $|0\rangle$ or $|1\rangle$ is the same after applying the phase shift gate since the amplitude of $|0\rangle$, $|1\rangle$ is left unchanged. Furthermore, the phase shift gate can be expressed as

$$P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix} = \frac{1}{2}(I + \sigma_z) + \frac{e^{i\varphi}}{2}(I - \sigma_z) = \frac{1 + e^{i\varphi}}{2}I + \frac{1 - e^{i\varphi}}{2}\sigma_z = |0\rangle\langle 0| + e^{i\varphi}|1\rangle\langle 1|$$

Here, we showcase the rotation gates along with their matrix representation and transformations.

1. Rotation around the x-axis is denoted by $R_X(\theta) = e^{-i\frac{\theta}{2}X}$, and its matrix is given by $\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$. The transformation of the gate on the qubit states is as follows

$$R_X(\theta) |0\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle - i\sin\left(\frac{\theta}{2}\right) |1\rangle$$

$$R_X(\theta) |1\rangle = \cos\left(\frac{\theta}{2}\right) |1\rangle - i\sin\left(\frac{\theta}{2}\right) |0\rangle$$

2. Similarly, rotation around the y-axis is denoted by $R_Y(\theta) = e^{-i\frac{\theta}{2}Y}$, and its matrix is given by $\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$. The transformation is

$$R_Y(\theta) |0\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + \sin\left(\frac{\theta}{2}\right) |1\rangle$$

$$R_Y(\theta) |1\rangle = \cos\left(\frac{\theta}{2}\right) |1\rangle - \sin\left(\frac{\theta}{2}\right) |0\rangle$$

3. Finally, rotation around the z-axis is denoted by $R_Z(\theta) = e^{-i\frac{\theta}{2}Z}$, and its matrix is given by $\begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$. The transformation of the matrix on the qubit states is

$$R_Z(\theta) |0\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle - i \sin\left(\frac{\theta}{2}\right) |0\rangle = \left(\cos\left(\frac{\theta}{2}\right) - i \sin\left(\frac{\theta}{2}\right)\right) |0\rangle$$

$$R_Z(\theta) |1\rangle = \cos\left(\frac{\theta}{2}\right) |1\rangle + i \sin\left(\frac{\theta}{2}\right) |1\rangle = \left(\cos\left(\frac{\theta}{2}\right) + i \sin\left(\frac{\theta}{2}\right)\right) |1\rangle$$

1.5 Two Qubit Gates

In addition to the gates mentioned in section 1.4, there also are two-qubit gates that operate on two qubits. These gates allow for the interaction and entanglement between the qubits. Typically, these gates have one control and one target qubit. The control qubit is the one that we examine to determine its state, and it is also the qubit that determines what state the target qubit will be in.

Firstly, the most essential two-qubit gate is the Controlled-NOT or CNOT Gate (CN). It takes two inputs and has two outputs. The CNOT gate acts like a classical NOT operation and flips the target qubit only when the control qubit is in the state $|1\rangle$. Otherwise, when the control qubit is $|0\rangle$, the target qubit remains the same. The matrix representation of the CNOT gate is shown below

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

By analyzing the CNOT matrix, we write it as a combination of Pauli Gates as follows

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & \sigma_x \end{bmatrix} = I \otimes I + \frac{I - \sigma_Z}{2} \otimes (\sigma_x - I)$$

The circuit representation of the gate is demonstrated in Figure 1.9

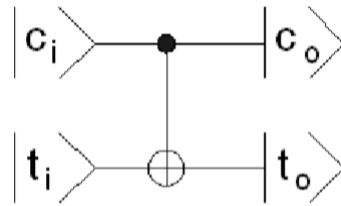


Figure 1.9: CNOT Gate.

The control qubit is denoted as $|c\rangle$ and the target qubit as $|t\rangle$. The CNOT gate can be written mathematically as

$$CNOT |c, t\rangle = |c, c \oplus t\rangle$$

For a more precise understanding of the CNOT gate, we present a table with the action of the CNOT gate (Table 1.4) and the truth table for the two-input XOR gate (\oplus) (Table 1.5).

$c_i t_i$	$c_o t_o$
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 01\rangle$
$ 10\rangle$	$ 11\rangle$
$ 11\rangle$	$ 10\rangle$

Table 1.4: Action of CNOT gate

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.5: Truth table of XOR gate

Secondly, there is the Controlled-Z or CZ gate that performs a phase flip on the target qubit if the control qubit is $|1\rangle$. Its matrix form is presented as follows

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The gate's action is illustrated in Table 1.6, which shows the input and output states of the control and target qubits. As you can see, if both qubits are in the state of $|11\rangle$, the output state of the target qubit will be flipped to $|-11\rangle$.

$c_i t_i$	$c_o t_o$
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 01\rangle$
$ 10\rangle$	$ 10\rangle$
$ 11\rangle$	$ -11\rangle$

Table 1.6: Action of CZ gate

The SWAP gate (S) is another two-qubit gate in quantum computing. Its function is to exchange the states of two qubits. After applying the SWAP gate, the state of the first qubit becomes the state of the second qubit, and vice versa. The SWAP gate is also reversible, which means that you can apply it again to restore the original states of the qubits. This gate is particularly useful when you need to rearrange the order of qubits in a quantum circuit or when you want to move information from one qubit to another. The SWAP gate can be represented by the matrix:

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The symbol of the SWAP gate is demonstrated in Figure 1.10. Table 1.7 shows the action of the SWAP gate on the qubits.

Note that the CNOT and the SWAP gates can also operate on three qubits. The three-qubit gates are an extension of the principles of the two-qubit gates. The Toffoli or CCNOT Gate (CCN) flips the target qubit if both control qubits are in the state $|1\rangle$,

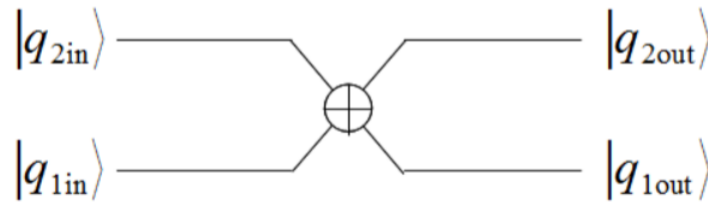


Figure 1.10: SWAP Gate symbol

$q_{2in}q_{1in}$	$q_{2out}q_{1out}$
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 10\rangle$
$ 10\rangle$	$ 01\rangle$
$ 11\rangle$	$ 11\rangle$

Table 1.7: Action of SWAP gate

similar to the CNOT Gate. You can see the representation of the gate in Figure 1.11. The Fredkin Gate, also known as the Controlled-SWAP gate, is a quantum gate that swaps the states of the second and third qubits if and only if the state of the first qubit is $|1\rangle$. The gate is shown in the figure 1.12

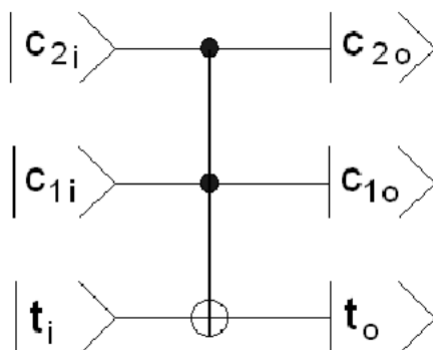


Figure 1.11: CCNOT Gate

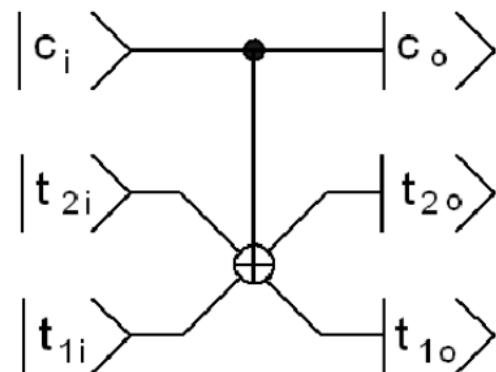


Figure 1.12: Fredkin Gate

The matrices corresponding to the CCNOT and Fredkin Gates are depicted below.

If we examine the matrices closely, we will observe that they can be decomposed into four smaller matrices representing known gates. Therefore, the CCNOT and Fredkin gates can be constructed from simpler gates as shown below

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$CCNOT = \begin{bmatrix} I_4 & O_4 \\ O_4 & CN \end{bmatrix} = I \otimes I_4 + \frac{I - \sigma_Z}{2} + (CN - I_4)$$

$$F = \begin{bmatrix} I_4 & O_4 \\ O_4 & S \end{bmatrix} = I \otimes I_4 + \frac{I - \sigma_Z}{2} + (S - I_4)$$

1.6 Quantum Entanglement

Entanglement is a unique property of multi-qubit state space, which is a multi-qubit system. It is a valuable resource of great utility that plays a key role in many applications of quantum computation and quantum information. The challenge with entanglement is that it is a uniquely quantum phenomenon, lacking any classical analog. It occurs when two or more particle-qubits become correlated in such a way that the quantum state needs both (or all) particle-qubits to be described. What's remarkable is that the states interact in a way that the state of one qubit directly impacts the state of the other, even if they are very far apart. So, no matter the distance separating the entangled particles-qubits, the system behaves as a whole. The measurement is no exception in this paradox behavior, and any measurement of a particle-qubit results in a collapse that instantly affects the probability amplitudes of the other entangled qubits, and it changes the original state. Einstein referred to this behavior as "spooky action at a distance" in 1935 when he, Boris Podolsky, and Nathan Rosen wrote a paper describing some of the strange implications of entanglement, known as the EPR paradox. This paper played a

crucial role in the development of quantum information theory but it quickly became a central piece in ongoing discussions. Almost three decades later, John S. Bell wrote a paper in 1964 introducing the famous four maximally entangled two-qubit Bell states. In general, Bell's research opened up the potential use of these super-strong correlations for communication purposes.

So, what exactly is entanglement? It's a unique concept that occurs when two qubits are so intricately linked that they cannot be described independently. In mathematical terms, this phenomenon is described as a state of two qubits that cannot be written as a tensor product. To delve deeper into this intriguing phenomenon, let's explore the creation of entanglement between two qubits using a simple example of quantum entanglement.

In this example, we start with a computational basis of $|00\rangle$. We use a Hadamard gate and a CNOT gate to create entanglement between these two qubits, as shown in Figure 1.13.

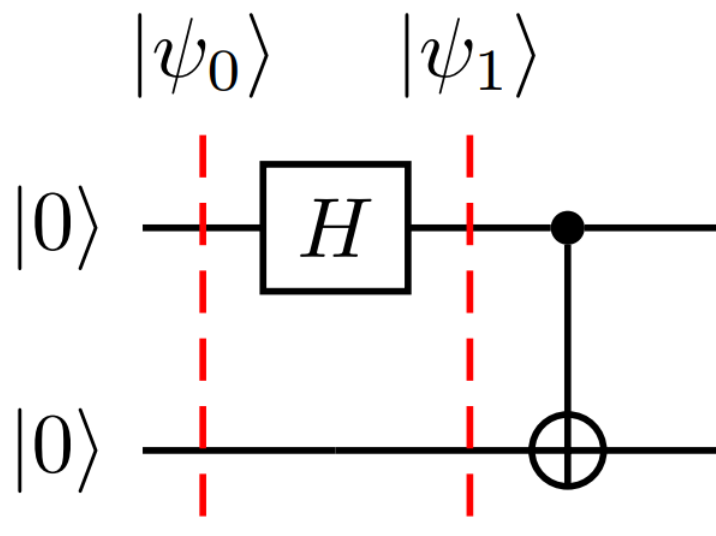


Figure 1.13: Quantum circuit to create Bell state

Firstly we have $|\psi_0\rangle = |00\rangle$. We apply a Hadamard gate to the first qubit while we do nothing on the second qubit (meaning we apply an Identity gate), resulting in the state

$$|\psi_1\rangle = H |\psi_0\rangle = (H \otimes I) |00\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) |0\rangle = \frac{|00\rangle + |10\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} (|00\rangle + |10\rangle)$$

Next, we apply a CNOT gate to our two qubits, with the first being the control and the second being the target qubit, and we get the state

$$|\psi_2\rangle = CNOT |\psi_1\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$$

This state that we just created is one of the four Bell states or EPR states, known as $|\beta_{00}\rangle$ and is given by:

$$|\beta_{00}\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Note how the state space cannot be decomposed into component spaces. The rest of the Bell states or EPR states can be formed by applying the other basic two-qubit inputs to the circuit ($|01\rangle$, $|10\rangle$, $|11\rangle$). It is fascinating to observe that by measuring the state of one qubit, we instantly gain knowledge about the state of the other qubit as well.

$$|\beta_{10}\rangle = \frac{1}{\sqrt{2}} (|00\rangle - |11\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

$$|\beta_{01}\rangle = \frac{1}{\sqrt{2}} (|01\rangle + |10\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$|\beta_{11}\rangle = \frac{1}{\sqrt{2}} (|01\rangle - |10\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \end{bmatrix}$$

In general, the mnemonic notation that the circuit transforms the input is given by the equation

$$|\beta(x, y)\rangle = \frac{|0y\rangle + (-1)^x |1.\bar{y}\rangle}{\sqrt{2}}$$

where $x \in \{0, 1\}$, $y \in \{0, 1\}$ and \bar{y} is the negation of y .

1.7 Deutsch- Jozsa algorithm

In 1992, David Deutsch and Richard Jozsa created the Deutsch-Jozsa algorithm, which stands as one of the earliest examples of a quantum algorithm. While this deterministic quantum algorithm does not have practical use, it does demonstrate how quantum circuits can outperform the capabilities of classical ones. In fact, it is exponentially faster than any possible deterministic classical algorithm!

The algorithm tackles the problem of a Boolean function f . For input, it takes a binary bit string of length n and produces a Boolean output of either a zero or a one for each input value.

$$f : \{0, 1\}^n \rightarrow \{0, 1\}$$

The goal of the algorithm is to determine if a function f is constant or balanced. By constant, we define a function that produces zero or one on all inputs, while by balanced, we expect an equal number of zeros and ones.

For example, for $f : \{0, 1\} \rightarrow \{0, 1\}$ we have 4 possibilities which can lead to the following two cases: a constant or balanced function, as shown in figure 1.14.

For the example we just demonstrated, it is obvious that a classical computer would need

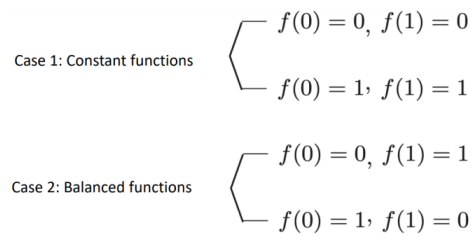


Figure 1.14: Two cases: Constant or Balanced function.

to evaluate the two functions separately to examine which of the two cases is our answer. More precisely, a classical computer would have to be used twice, once to evaluate $f(0)$ and once to evaluate $f(1)$, and then store the answers somewhere in order to, finally, compare them. Thus, in general, a classical computer will need to evaluate, at best, half, plus one, of the inputs, $\frac{2^n}{2+1}$ queries, to be certain if the function is balanced or constant, which means that the worst-case scenario requires $O(n)$. However, quantumly, with the Deutsch-Jozsa algorithm, the problem can be solved with a single evaluation of f (i.e., one query) by using a black box.

The black box performs the transformation $|x\rangle |y\rangle \rightarrow |x\rangle |y \oplus f(x)\rangle$ for $x \in \{0, \dots, 2^n - 1\}$ and $f(x) \in \{0, 1\}$. (figure 1.15)

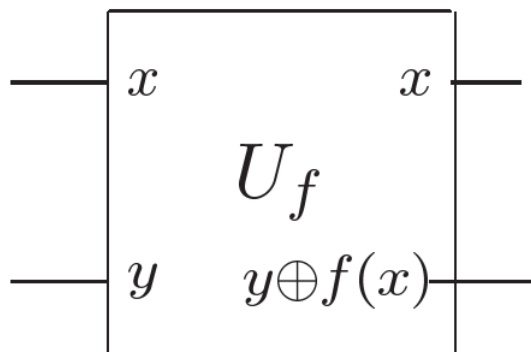


Figure 1.15: Black Box

The operation gate U is called an oracle, but it can be thought of as a black box. We use it to pass information to all qubits simultaneously, with the values of $f(x)$ for each x being the information. Gate U must be represented by a unitary matrix in order to be considered valid; here the gate is a $2^{n+1} \times 2^{n+1}$ matrix acting on $n + 1$ qubits.

1.7.1 The Algorithm

In this section, we will dive into the Deutsch-Jozsa quantum algorithm and will discuss the algorithmic steps in detail. The steps of the algorithm are depicted in Figure 1.16. Let us trace the states through this circuit.

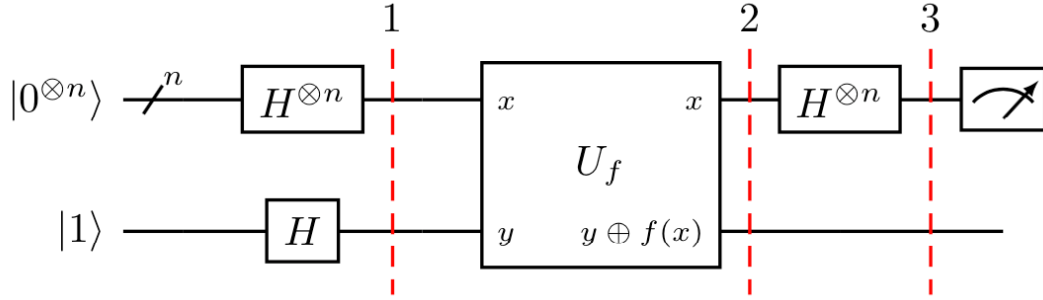


Figure 1.16: Deutsch- Jozsa algorithm quantum circuit

When we write that we apply the $H^{\otimes n}$ transform in the first input, it means that we apply the Hadamard gate to each of the n qubits. The transformation goes as follows

$$H^{\otimes n} |x_1, \dots, x_n\rangle = \sum_{z_1, \dots, z_n} \frac{(-1)^{x_1 \cdot z_1 + \dots + x_n \cdot z_n}}{\sqrt{2^n}} |z_1, \dots, z_n\rangle$$

This can be summarized as

$$H^{\otimes n} |x\rangle = \sum_z \frac{(-1)^{x \cdot z}}{\sqrt{2^n}} |z\rangle$$

where $x \cdot z$ is the bitwise inner product of x and z , modulo 2, with output one, when there is an odd number of $x_i z_i$ and output zero otherwise.

The input state is

$$|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$$

The first step is to apply the Hadamard gate to the $n + 1$ qubits. The transformation of the input state is as follows

$$\begin{aligned}
|\psi_1\rangle &= (H^{\otimes n} \otimes H) |\psi_0\rangle = (H^{\otimes n} \otimes H) |0\rangle^{\otimes n} |1\rangle = H^{\otimes n} |0\rangle^{\otimes n} \otimes H |1\rangle = \\
&= \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{1.1}
\end{aligned}$$

Note how the result of this transformation is a superposition of all possible values for the first input and an evenly weighted superposition of zero and one for the second input.

$$|\psi_1\rangle = \sum_{x \in \{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$$

Next, we apply the oracle on $|\psi_1\rangle$ with U acting $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. The result of this operation is $|\psi_2\rangle$, which can be expressed as follows:

$$|\psi_2\rangle = U |\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |x\rangle (|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle)$$

$$|\psi_2\rangle = \sum_{x \in \{0,1\}^n} \frac{(-1)^{f(x)} |x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$$

Finally, now that we calculated the function f using the oracle U , we apply another Hadamard $H^{\otimes n}$ transform to obtain the $|\psi_3\rangle$, which can be written as

$$|\psi_3\rangle = H^{\otimes n} |\psi_2\rangle = \sum_{z \in \{0,1\}^n} \frac{(-1)^{x \cdot z}}{\sqrt{2^n}} |z\rangle \sum_{x \in \{0,1\}^n} \frac{(-1)^{f(x)}}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$$

$$|\psi_3\rangle = \sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} \frac{(-1)^{x \cdot z + f(x)}}{2^n} |z\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$$

What follows next is the measurement to obtain the final output. We will leave the last qubit out since it is not being measured. From now on the term $\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$ from $|\psi_3\rangle$ will be ignored. We should also note that the amplitude for all zero state $|0\rangle^{\otimes n}$ from $|\psi_3\rangle$ is $\sum_{x \in \{0,1\}^n} \frac{(-1)^{f(x)}}{2^n}$. Let us look at the two possible cases to discern what happens.

In the case where $f(x)$ is constant, the probability amplitude for $z = |0\rangle^{\otimes n}$ is

expressed as

$$\sum_{x \in \{0,1\}^n} \frac{1}{2^n} = \pm 1$$

depending on the constant value of $f(x)$. This is because we know that $z = |0\rangle^{\otimes n}$ and since z is equal to zero, the product $x \cdot z$ must be equal to zero too. Then the term

$$(-1)^{x \cdot z + f(x)}$$

is either -1 or $+1$ for all values of x , given that the $f(x)$ is constant and equal to zero or one for all the possible x . Hence, in this case, an observation yields zeros for all qubits when measured.

On the other hand, if $f(x)$ is balanced then the probability of obtaining $z = |0\rangle^{\otimes n}$ is expressed as

$$\sum_{x_1 \in \{0,1\}^n} \frac{+1}{2^n} + \sum_{x_2 \in \{0,1\}^n} \frac{-1}{2^n} = 0$$

Again we have $z = |0\rangle^{\otimes n}$ and $x \cdot z$ is equal to zero, but this time, the term

$$(-1)^{x \cdot z + f(x)}$$

for certain values of x will be $+1$, while for others, it will be -1 . This occurs when $f(x)$ is perfectly balanced, with half of all possible x resulting in a value of one and the other half resulting in zero. When this balance is achieved, the positive and negative contributions to the amplitude cancel each other out for the state $|0\rangle^{\otimes n}$, resulting in an amplitude of zero. Therefore, upon observation, the outcome will not be zero for at least one qubit.

To summarize, when run on a quantum computer, the Deutsch-Jozsa algorithm can solve the problem of determining whether a function is constant or balanced with exactly one evaluation. If the measurements show all zeros, then the function is constant; otherwise, if we measure anything except zero, then the function is balanced.

Chapter 2

Optimization: from classical to quantum

Quantum computers can improve certain processes, such as optimization problems, more efficiently than classical computers. These problems are commonly found in industrial areas such as finance, logistics, and materials science. In this chapter, we will see how to apply quantum algorithms to solve optimization problems. Generally, there are two main approaches for quantum algorithms: quantum annealing and gate-based designs. The first is based on adiabatic quantum computation using the adiabatic theorem of quantum mechanics, and is a method used to find the optimal solution to problems involving a large number of solutions by finding a minimum energy state of a given Hamiltonian. Quantum annealers can be effective for certain optimization problems (problem-dependent), but they have limitations and are incapable of universal quantum computation due to their hardware construction. In contrast, in gate-based approaches, one has access to universal-type gates, which offer the potential to solve a wide range of quantum problems, regardless of the specific problem (problem independent). However, gate-based computers also encounter challenges related to hardware limitations and the accurate implementation of gates due to the interaction of quantum systems with the environment.

Generally, optimization problems are mathematical problems where the goal is to find the best solution from a set of possible solutions. Such problems search for a solution

that minimizes or maximizes an objective function. There are two groups of optimization problems depending on whether the variables are continuous or discrete. In the next section (2.1) we will see more about the discrete optimization problems (also known as combinatorial optimization problems) that use a method to find their optimal solution from a finite but large set of discrete solutions. More specifically, we would focus on solving binary optimization problems where the variables can only be binary and involve finding "optimal" bit-strings composed of 0's and 1's among a finite set of bit-strings.

These types of problems, especially the ones with many variables are, in general, time-consuming and computationally expensive. More precisely, combinatorial optimization problems belong to the NP-hard class of optimization problems, which require exponential time to be solved to optimality. The typical classical optimization algorithms, which search through the space of all possible solutions and evaluate each solution until the best one is found, can be inefficient. So, the idea is to seek a good enough solution instead of a perfect one. For that reason, heuristics and approximation algorithms are introduced. These algorithms, which determine near-optimal and approximate solutions, respectively, can reduce the time complexity and speed up the process considerably. However, reaching a solution can still take a long time, especially for large instances of the problem.

2.1 Combinatorial Optimization

Combinatorial optimization problems are defined by n bits (binary variables) and m clauses. Every clause represents a constraint on a subset of the bits. These constraints are satisfied for certain assignments of the bits and unsatisfied for others. Defined on n bit strings, the objective function is the number of satisfied clauses,

$$c(z) = \sum_{a=1}^m c_a(z)$$

where $z = z_1 z_2 \dots z_n$ with $z_i \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}$ is the bit string, $c_a(z) = 1$ if z satisfies clause a and $c_a(z) = 0$ otherwise. c_a usually depends on only a few of the n bits. The goal is to maximize the objective function by finding the assignment of z that

satisfies the largest number of clauses. To achieve approximate optimization, we must search for a string z that results in a value of $c(z)$ that is close to the maximum value of c .

2.2 Quadratic Unconstrained Binary Optimization (QUBO)

In the field of computer science, QUBO problems are frequently utilized. These problems involve variables that have the potential to take on two values: 1 (TRUE) or 0 (FALSE). The Quadratic Unconstrained Binary Optimization problem (QUBO) has become a unifying model for representing various combinatorial optimization problems. The QUBO model is frequently used to solve a wide range of optimization problems, including, but not limited to, max-cut, graph coloring, and various machine learning problems.

Definition 2.2.1. Quadratic Unconstrained Binary Optimization (QUBO) is a type of discrete combinatorial optimization problem that aims to find the optimal bit-string x^* , represented by a series of 0s and 1s, which minimizes or maximizes a given cost function. That is

$$x = \min_x C_Q(x)$$

A QUBO is often represented as an undirected graph where each binary variable is represented as a node in the graph. The coefficients of the linear terms are assigned to each node, and the coefficients of the quadratic terms are assigned as the weighted edge between two nodes.

Given that the graph is $G = [N, E]$ with node set $N = \{1, 2, \dots, i, \dots, n\}$ and edge set $E = \{(i, j) : i, j \in N\}$. We define QUBO as:

$$C_Q(x) = \sum_{(i,j) \in E} q_{ij} x_i x_j + \sum_{i \in N} c_i x_i \text{ subject to } x_i = \{0, 1\} \text{ where } i \in N$$

with the quadratic coefficients $q_{ij} \in \mathbb{R}$, the weight of edge (i, j) , and the linear coefficients $h_i \in \mathbb{R}$ of the binary variables x_i .

The equivalent definition with the coefficients mentioned above represented into an upper-diagonal matrix formulation is:

$$C_Q(x) = x^t Q x : x \in \{0, 1\}^n$$

where $Q \in \mathbb{R}^{n \times n}$ is an n-by-n square symmetric matrix of coefficients q_{ij} (real weights).

To illustrate the above, let us consider a specific example. For instance, if you have 3 nodes, the expansion from matrix formulation to the cost function can be explained as follows

$$C_Q(x) = x^t Q x \text{ with } A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \text{ and } x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \text{ so}$$

$$C_Q(x) = x^t Q x = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Note that the matrix A is symmetric, so the elements $a_{ij} = a_{ji}$

$$\begin{aligned} C_Q(x) &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ &= a_{12}x_1x_2 + a_{13}x_1x_3 + a_{23}x_2x_3 + a_{11}x_1x_1 + a_{22}x_2x_2 + a_{33}x_3x_3 \end{aligned}$$

Note that since the variables are binary x_i can only take values 0 or 1, so $x_i^2 = x_i$ and we can write

$$C_Q(x) = a_{12}x_1x_2 + a_{13}x_1x_3 + a_{23}x_2x_3 + a_{11}x_1 + a_{22}x_2 + a_{33}x_3$$

And so we get

$$C_Q(x) = \sum_{(i,j) \in E} q_{ij}x_i x_j + \sum_{i \in N} c_i x_i \text{ with } q_{ij} = 2a_{ij} \text{ and } c_i = a_{ii}$$

In the case of QUBO, it is apparent that the complexity grows exponentially with 2^n potential solutions to examine. Moreover, as the problem size increases by one, the

number of potential solutions doubles. In general, QUBO is an NP-hard problem, and it quickly becomes computationally infeasible. For this reason, classical methods to find optimal solutions have been designed, such as the CPLEX, Gurobi, and SCIP, but even when such methods are used, it can take days to produce high-quality solutions, sometimes with no success. Fortunately, metaheuristic methods like Path relinking [19], Tabu search [16] [17], evolutionary algorithms [21], and Simulated Annealing [24] are achieving remarkable success by finding high-quality solutions in a predetermined time, although they may not be optimal.

We will show a simple example for clearer comprehension. Consider the optimization problem:

$$\text{Minimize } y = -5x_1 - 3x_2 - 8x_3 + 4x_1x_2 + 8x_1x_3 + 2x_2x_3$$

where the variables x_1, x_2, x_3 are binary and they represent our decision choices. The function y to be minimized is a quadratic function with binary variables. Notice how the function has both linear $-5x_1 - 3x_2 - 8x_3$ and quadratic $4x_1x_2 + 8x_1x_3 + 2x_2x_3$ components. By keeping in mind the property $x_i^2 = x_i$ we write $-5x_1^2 - 3x_2^2 - 8x_3^2$ and then we transform this problem into a matrix form, as shown below:

$$\text{Minimize } y = [x_1 \ x_2 \ x_3] \begin{bmatrix} -5 & 2 & 4 \\ 2 & -3 & 1 \\ 4 & 1 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

With this matrix notation, we represent the QUBO problem as: $\text{Minimize } y = x^t Q x$. Here, the matrix Q encapsulates all the problem data and provides an alternative to traditional constrained problem representations. Since we have a 3x3 QUBO problem, we have a 3-bit solution.

We present a heat-map visualization of the QUBO matrix to help understand the structure of the matrix and the relationships between the variables in the problem (see figure 2.1). Each cell in the heat map corresponds to an element in the QUBO matrix; the cell's color represents that element's value. The color scale on the figure's right shows

that darker colors represent lower values, and lighter colors represent higher values.

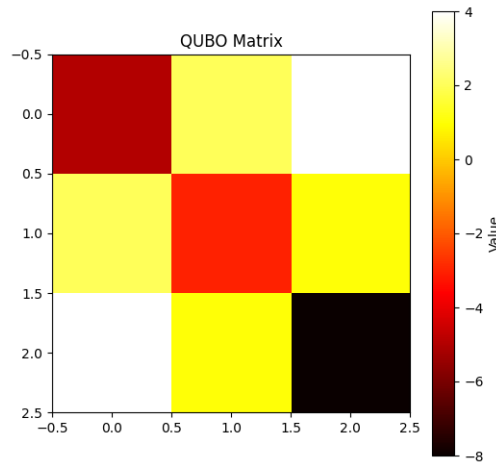


Figure 2.1: Heat-map visualization of the QUBO matrix

The optimal solution to the above problem is: $y = -9$ for $x = (0, 1, 1)$ (figure 2.2).

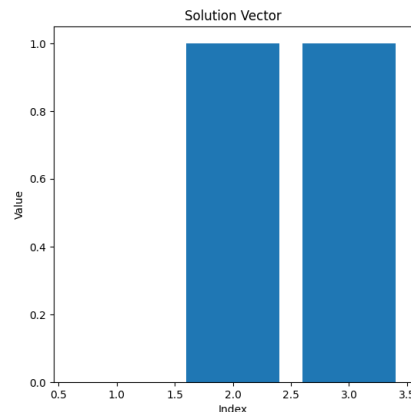


Figure 2.2: Bar chart of the solution vector, indices 1.0, 2.0, 3.0 represent our 3-bit solution

As mentioned so far, the Quadratic Unconstrained Binary Optimization model does not contain any constraints. However, in reality, a large number of problems require some constraints to be satisfied. So, the QUBO model is being reformulated to ensure all constraints are satisfied while the optimizer searches for good solutions. The proposed method introduces quadratic penalties into the objective function based on the original constraint. The idea of the penalties is that when the solution is feasible, and the

constraints are satisfied, the penalty is equal to zero, and the objective function is minimized; otherwise, the penalty is equal to some positive number. Here, we present a table 2.1 of some known in-advance penalties; notice how all variables are binary. It is important to note that the parameter P is a large enough positive value.

Classical Constraints	Equivalent Penalty
$x + y \leq 1$	$P(xy)$
$x + y \geq 1$	$P(1 - x - y + xy)$
$x + y = 1$	$P(1 - x - y + 2xy)$
$x \leq y$	$P(x - xy)$
$x = y$	$P(x + y - 2xy)$
$x_1 + x_2 + x_3 \leq 1$	$P(x_1x_2 + x_1x_3 + x_2x_3)$

Table 2.1: Table of known constraint/penalty pairs.

2.2.1 From QUBO to Hamiltonian

Before we continue this chapter, let us denote a valuable definition.

Informal definition. In quantum mechanics, the Hamiltonian, a mathematical concept introduced in 1835 by Sir William Rowan Hamilton, is an operator corresponding to the total energy of a given quantum system described by Hermitian matrix $H = H^\dagger$. The energy of a system in the state $|\psi\rangle$ is given by expectation value

$$E(|\psi\rangle) = \langle\psi|H|\psi\rangle$$

The ground state is defined as the lowest energy state $|\psi^*\rangle$ of a quantum system

$$|\psi^*\rangle = \arg \min_{|\psi\rangle \in \mathcal{H}} E(|\psi\rangle)$$

A Hamiltonian for a quantum system can also be defined as a function that maps energies to certain states, known as eigenstates.

We want to formulate QUBO instances in a Hamiltonian problem and so to set the QUBO bit-string in such a way that it can be “identified” with a computational basis state of a quantum system. We mentioned that the QUBO cost function is

$$C_Q(x) = \sum_{(i,j) \in E} q_{ij} x_i x_j + \sum_{i \in N} c_i x_i$$

and we will prove that the Hamiltonian operator that encodes the cost function $C_Q(x)$ is

$$H = \sum_{i,j=1}^n \frac{1}{4} q_{ij} z_i z_j - \sum_{i=1}^n \frac{1}{2} \left(c_i + \sum_{j=1}^n q_{ij} \right) z_i + \left(\sum_{i,j=1}^n \frac{q_{ij}}{4} + \sum_{i=1}^n \frac{c_i}{2} \right)$$

But why is the formulation of QUBO instances into Hamiltonian important? It is because it provides a crucial bridge between the QUBO problems and quantum computing, which helps us to solve optimization problems.

We have for the Pauli-Z gate $\sigma_i^z = z_i$ that we apply:

$$z_i |x\rangle = (-1)^{x_i} = 1 - 2x_i \Rightarrow$$

$$x_i = \frac{1 - z_i}{2} |x\rangle$$

So from the above and the cost function we have for the Hamiltonian is

$$H = \sum_{i,j=1}^n q_{ij} \frac{1 - z_i}{2} \frac{1 - z_j}{2} + \sum_{i=1}^n c_i \frac{1 - z_i}{2} =$$

$$= \sum_{i,j=1}^n \frac{q_{ij}}{4} + \sum_{i=1}^n \frac{c_i}{2} - \sum_{i=1}^n \left(\sum_{j=1}^n \frac{q_{ij}}{4} \right) z_i - \sum_{j=1}^n \left(\sum_{i=1}^n \frac{q_{ij}}{4} \right) z_j - \sum_{i=1}^n \frac{c_i}{2} z_i + \sum_{i,j=1}^n \frac{q_{ij}}{4} z_i z_j$$

$$= \sum_{i,j=1}^n \frac{q_{ij}}{4} z_i z_j - \sum_{i=1}^n \left(\frac{c_i}{2} + \sum_{j=1}^n \frac{q_{ij}}{4} \right) z_i + \sum_{i,j=1}^n \frac{q_{ij}}{4} + \sum_{i=1}^n \frac{c_i}{2}$$

We managed successfully to represent the Hamiltonian encoded with the QUBO cost function $C_Q(x)$ with the terms of the equation corresponding to the quadratic and linear term of the cost function. Note that the extra terms are too used to define collectively the energy of the QUBO problem.

2.3 Ising Model

In this section, we will describe a classical mathematical model of statistical mechanics called the Ising model and its connections to QUBO. Named after the German physicist Ernst Ising and his professor Wilhelm Lenz, the Ising model is used to describe the magnetic interactions between spins and their behavior in a system. More specifically, this widely studied model consists of discrete variables representing magnetic dipole moments of atomic "spins", which are capable of existing in one of two states. The neighboring pairs of spins that interact with each other are usually arranged in a (square) n -dimensional lattice of sites (see figures 2.3, 2.4). We usually label each site with an index i , and we call the two states $+1$ and -1 . We symbolize with $\sigma_i = -1$, the i 'th site that is in the state -1 .

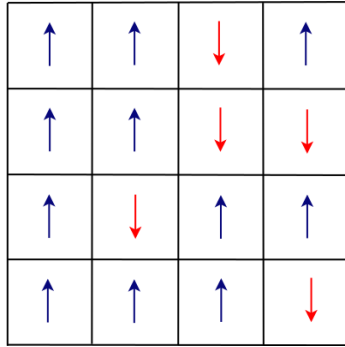


Figure 2.3: A lattice of sites where blue sites are in state $\sigma=+1$ and red sites are in state $\sigma=-1$.

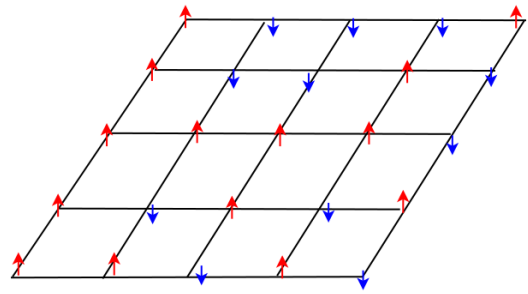


Figure 2.4: A lattice from a different angle.

One of the physical systems that the Ising model can represent is a magnet. Each site represents a particular spin, and each spin can be thought of as a mini magnet. If the spin points up ("spin up" \uparrow), we represent it with $\sigma_i = +1$; otherwise, if the spin points down ("spin down" \downarrow), we represent it with $\sigma_i = -1$. Notice that if all the spins are aligned, then the whole lattice of spins behaves as a big magnet.

The Hamiltonian of this system, which represents the energy of every possible configuration of the spins in the magnet, is:

$$H = \sum_{(i,j)} J_{ij} \sigma_i \sigma_j + \sum_i h_i \sigma_i$$

where h_i represents the external magnetic field on spin i that indicates whether it is spin up or spin down, J_{ij} represents the coupling strength between spin i and j that indicates whether neighbors prefer to align (ferromagnetic) or to anti-align (anti-ferromagnetic), and the $\sigma_i\sigma_j$ are the individual spins on each of the lattice sites. It is worth noting the duality of the spin Hamiltonian; the first summation term in the spin Hamiltonian represents the interactions between spins (pairs/bonds of neighboring lattice sites), while the second summation term represents the external field trying to align all the spins in a single direction.

Like in many problems, the objective is to minimize the Hamiltonian and, thus, find the minimum energy. Note that local optima may exist in the energy configuration, so finding the global optima -i.e. the solution, can be challenging.

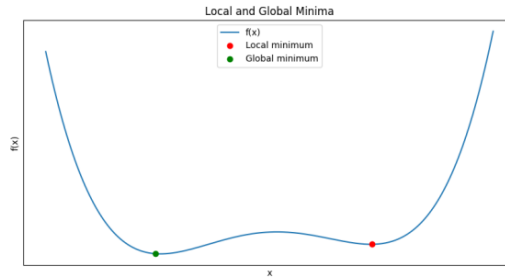


Figure 2.5: Global and local minimum

2.3.1 Ising Model and QUBO Model

The early application area of quantum computing has been optimization, mainly due to the ability to map many problem instances onto Ising problems, which can in turn be mapped onto a quantum computer. QUBO and Ising problems are closely connected. In fact, a QUBO problem can be transformed into a simple Ising problem through a linear transformation (one without an external magnetic field). Let us see how to transform the classical Ising model into its quantum counterpart.

The Ising model and QUBO are equivalent via a linear transformation of the variables. As it was mentioned before, the variables of the classical Ising Hamiltonian are the spins $\sigma_i = \pm 1$. As for the quantum mechanics world, the classical variables are represented by linear operators. Thus, the transformation from classical to quantum occurs by the

replacement of σ_i integers variables with the quantum operators σ_i^z . We have

$$H = H_Q = \sum_{(i,j)} J_{ij} \sigma_i^z \sigma_j^z + \sum_i h_i \sigma_i^z$$

where $\sigma_i^z \in \{-1, 1\}$ is the Pauli-z operator acting on the i^{th} spin.

The reason why the Z-operator was chosen is because the variables ± 1 are also the eigenvalues of that operation, and so we get the values of σ_i as probability amplitudes in the states where the Hamiltonian operator will act.

As we mentioned before QUBO's cost function is $C_Q(x) = x^t Q x$ where $x \in \{0, 1\}$ while the energy of the Ising model is $H(s) = \sigma^t J \sigma$ where $\sigma \in \{-1, 1\}$. The mapping between the binary variables $\{0, 1\}$ and the spins $\{-1, 1\}$ is done by substituting

$$x = \frac{(\sigma + 1)}{2}$$

.

We will show what a transformation of a QUBO into an equivalent Ising looks like. To do that, let's consider the example we solved in the QUBO chapter. We have

$$\text{Minimize } y = -5x_1 - 3x_2 - 8x_3 + 4x_1x_2 + 8x_1x_3 + 2x_2x_3$$

We start with our QUBO graph that we created to visualize our problem (Figure 2.6).

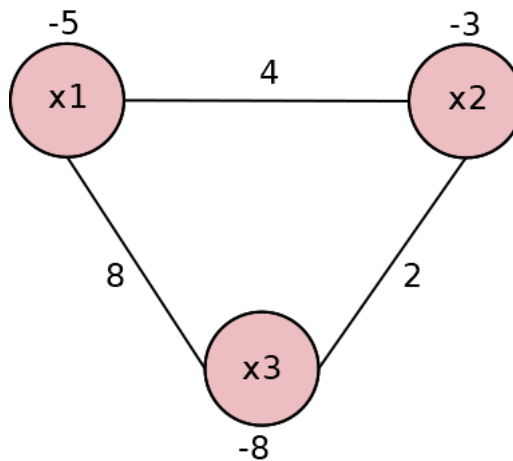


Figure 2.6: QUBO graph of our problem

What we want to do is to transform the QUBO model into an Ising model, which is also the process that happens when we run the problem on a QPU. As we just mentioned, we transform the problem from one with binary values $\{0,1\}$ to one that uses a set of variables that take the values $\{-1,+1\}$.

Recall that to map the binary variables we use the transformation $x_i = \frac{(\sigma_i+1)}{2}$. So we substitute the term into the QUBO equation.

We will start with the linear terms and we have

$$-5x_1 = -5 \frac{(\sigma_1 + 1)}{2} = -\frac{5}{2}(\sigma_1 + 1) = -2,5\sigma_1 - 2,5$$

$$-3x_2 = -3 \frac{(\sigma_2 + 1)}{2} = -\frac{3}{2}(\sigma_2 + 1) = -1,5\sigma_2 - 1,5$$

$$-8x_3 = -8 \frac{(\sigma_3 + 1)}{2} = -\frac{8}{2}(\sigma_3 + 1) = -4\sigma_3 - 4$$

For the quadratic terms we have

$$4x_1x_2 = 4 \frac{(\sigma_1 + 1)}{2} \frac{(\sigma_2 + 1)}{2} = 4 \left(\frac{\sigma_1\sigma_2 + \sigma_1 + \sigma_2 + 1}{4} \right) = \sigma_1\sigma_2 + \sigma_1 + \sigma_2 + 1$$

$$8x_1x_3 = 8 \frac{(\sigma_1 + 1)}{2} \frac{(\sigma_3 + 1)}{2} = 8 \left(\frac{\sigma_1\sigma_3 + \sigma_1 + \sigma_3 + 1}{4} \right) = 2\sigma_1\sigma_3 + 2\sigma_1 + 2\sigma_3 + 2$$

$$2x_2x_3 = 2 \frac{(\sigma_2 + 1)}{2} \frac{(\sigma_3 + 1)}{2} = 2 \left(\frac{\sigma_2\sigma_3 + \sigma_2 + \sigma_3 + 1}{4} \right) = 0,5\sigma_1\sigma_3 + 0,5\sigma_1 + 0,5\sigma_2 + 0,5$$

We combine all the terms that we calculate and we get the equivalent problem in the Ising model (we do not write the constant terms).

$$y = 0,5\sigma_1 - 0\sigma_2 - 1,5\sigma_3 + \sigma_1\sigma_2 + 2\sigma_1\sigma_3 + 0,5\sigma_2\sigma_3$$

Finally, we show the Ising graph.

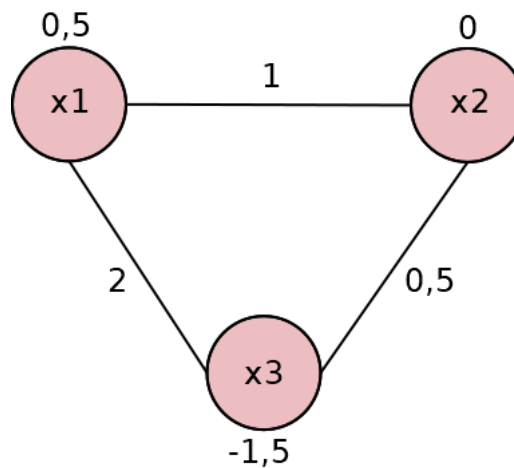


Figure 2.7: Ising graph of our problem

2.3.2 Solving MaxCut Problem - Hamiltonian

We will use the well known MaxCut problem as an example to explore how to translate a classical problem into a quantum Ising Hamiltonian.

The MaxCut problem takes as input a graph $G = (V, E)$, which is characterized by a set of n nodes V and a set of m undirected edges E (Figure 2.8). The task is to divide the nodes into two distinct sets while maximizing the number of edges that cross these sets.

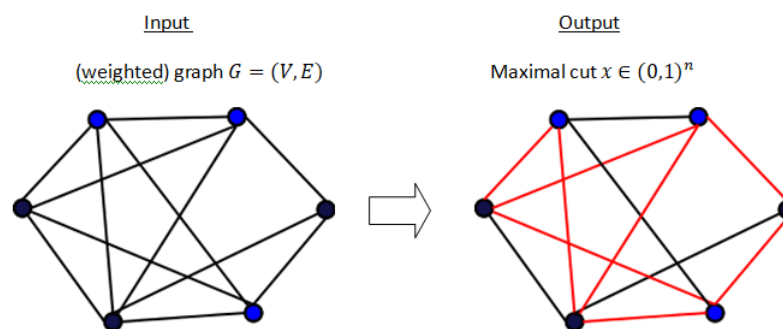


Figure 2.8: The dark blue and the light blue nodes are the two sets. In the output, with red color, are the edges that are cut.

For every node in the graph, there exists a binary variable $x_i \in \{0, 1\}$ that indicates which set the node belongs to. The objective function is made up of a term for each

edge in the graph. If the nodes of an edge have the same value, the term is 0; otherwise, it is 1. Therefore, the optimal solution will be a maximal cut of the graph G .

The MaxCut can be formulated as an optimizer problem

$$\max_s \frac{1}{2} \sum_{i,j \in E} (1 - s_i s_j) \text{ with } s_{i,j} \in \{-1, +1\}$$

We can see that if $s_{i,j}$ are the same sign, meaning no edge is cut, then $\frac{1}{2}(1 - s_i s_j) = 0$ and there is no contribution to the objective and if $s_{i,j}$ are the different sign, meaning an edge is cut, then $\frac{1}{2}(1 - s_i s_j) = 1$ and there is contribution to the objective.

As we mentioned in the previous section, to solve an optimization problem on a quantum computer, we need to convert the maximizing objective

$$\max_s C(z) = \frac{1}{2} \sum_{i,j \in E} (1 - s_i s_j)$$

into a problem of characterizing a quantum Hamiltonian H . So the solution to the problem $s \in \{-1, +1\}$ is now coming from the highest energy eigenstate $|s\rangle$.

To encode the objective into a quantum computer, we have to characterize a Hamiltonian H . The Hamiltonian is diagonal, with each value on the diagonal encoding as an eigenstate of the Hamiltonian and corresponding to the values of the objective function.

$$H = \begin{pmatrix} C(0\dots 00) & 0 & \dots & 0 & 0 \\ 0 & C(0\dots 01) & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & C(1\dots 10) & 0 \\ 0 & 0 & \dots & 0 & C(1\dots 11) \end{pmatrix}$$

Note how the Hamiltonian is too large to construct explicitly, so instead, we develop compact formulations and construct the H operator from basic building blocks -i.e. the Pauli operators. Let's also see how the Hermitian operator H affects a state.

$$H|0\dots 00\rangle = \begin{pmatrix} C(0\dots 00) & 0 & \dots & 0 & 0 \\ 0 & C(0\dots 01) & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & C(1\dots 10) & 0 \\ 0 & 0 & \dots & 0 & C(1\dots 11) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \Rightarrow$$

$$H|0\dots 00\rangle = C(0\dots 00)|0\dots 00\rangle$$

So, this Hamiltonian encodes the value of the function $C(x)$ when it acts on the computational basis state. Let's replace $|0\dots 00\rangle$ with the state $|x\rangle$.

$$H|x\rangle = \begin{pmatrix} C(0\dots 00) & 0 & \dots & 0 & 0 \\ 0 & C(0\dots 01) & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & C(1\dots 10) & 0 \\ 0 & 0 & \dots & 0 & C(1\dots 11) \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \Rightarrow$$

$$H|x\rangle = C(x)|x\rangle \quad \forall x \in \{0, 1\}^n$$

Back to the MaxCut objective, to construct the Hamiltonian, we map the binary variables s_i onto the eigenvalues of Z .

$$H = \frac{1}{2} \sum_{i,j \in E} (1 - Z_i Z_j)$$

What we have is $x_i = \frac{(1-s_i)}{2}$ with $s_i \in \{-1, +1\}$ and so when s_i is equal to one then the value x_i is zero, otherwise when s_i is equal to minus one one then the value x_i is one.

Let's consider the Pauli's Z -operator

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

and note that the Z -operator has eigenvalues $-1, +1$ with the eigenvectors being computational basis states.

$$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = (+1)|0\rangle$$

$$Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = (-1)|1\rangle$$

This can be generalized to

$$Z|x\rangle = (-1)^x|x\rangle \quad x \in \{0,1\}$$

We generalize the Z -operation action on multiple qubits; here is acting on the i -th qubit:

$$Z_i|x_0 \dots x_n\rangle = (I \otimes \dots \otimes Z_i \otimes \dots \otimes I)|x_0 \dots x_n\rangle = (-1)^{x_i}|x_0 \dots x_n\rangle$$

$$\text{with } x_i \in \{0,1\}, i = 1, \dots, n$$

The Z -operation acting on the i -th and j -th(adjacent) qubit:

$$Z_i Z_j|x_0 \dots x_n\rangle = I \otimes \dots \otimes Z_i \otimes Z_j \otimes \dots \otimes I|x_0 \dots x_n\rangle = (-1)^{x_i}(-1)^{x_j}|x_0 \dots x_n\rangle$$

So the MaxCut objective can additionally be written as:

$$\max_x \frac{1}{2} \sum_{i,j \in E} (1 - (-1)^{x_i}(-1)^{x_j}), \quad x_i \in \{0,1\}$$

This formulation is also helpful in verifying the MaxCut Hamiltonian as we see below

$$H|x\rangle = \frac{1}{2} \sum_{i,j \in E} (I - Z_i Z_j)|x_0 \dots x_n\rangle =$$

$$\begin{aligned}
&= \frac{1}{2} \sum_{i,j \in E} (|x_0 \dots x_n\rangle - Z_i Z_j |x_0 \dots x_n\rangle) \\
&= \frac{1}{2} \sum_{i,j \in E} (1 - (-1)^{x_i} (-1)^{x_j}) |x_0 \dots x_n\rangle = C(x) |x\rangle \\
&\quad \text{with } x_i \in \{0,1\}, i = 1, \dots, n
\end{aligned}$$

2.4 Quantum Adiabaticity

Adiabatic invariants have a long history. Einstein was the first to call the adiabatic hypothesis in 1911. Later, in the early 1910s, Paul Ehrenfest formulated and applied his adiabatic hypothesis, while the more modern version of adiabatic was suggested by Max Born and Vladimir Fock in 1928. Adiabatic theorem's original form was stated as follows:

Definition 2.4.1. A physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian spectrum.

Definition 2.4.2. If we have a time-varying Hamiltonian, $H(t)$ which is initially H_0 at $t = 0$, and subsequently H_1 at some later time, $t = \tau$, then if the system is initially in the ground state of H_0 , and as long as the time evolution of the Hamiltonian is sufficiently slow, the state is likely to remain in the ground state throughout the evolution, therefore being in the ground state of H_1 at $t = \tau$.

Suppose that we have a Hamiltonian H_1 whose ground state encodes the solution of the problem and an initial Hamiltonian H_0 . We begin in Hamiltonian H_0 , whose ground state can be prepared easily. We then slowly evolve our Hamiltonian, i.e., using a unitary operator, by adjusting the magnetic field which acts on qubits, given the time-dependent Hamiltonian

$$H(t) = \left(1 - \frac{t}{T}\right) H_0 + \frac{t}{T} H_1$$

where $t \in [0, \tau]$ and τ is the total evolution time. The unitary evolution operator is

$$U = e^{-i\frac{Ht}{\hbar}}$$

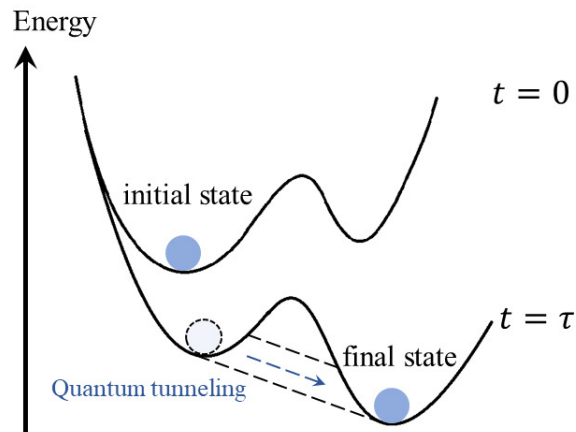


Figure 2.9: Quantum adiabatic evolution in quantum annealing. If the evolution time (τ) is long enough, the Hamiltonian that controls the system will change slowly, which will allow the system to transition to a new ground state using the quantum tunneling effect.

2.4.1 Quantum Annealing

Quantum annealers have been proved a good method to solve optimization problems by returning low-energy solutions naturally. However, we should keep in mind that modern-day quantum annealers are still highly vulnerable to noise, and it is quite challenging to scale them up. They also necessitate a considerable amount of energy to function and have limited connectivity.

Let us delve into how quantum annealing works in the D-Wave. The D-Wave QPU is composed of superconducting loops, which have qubits as their lowest energy states. These qubits have a magnetic field and a circulating current associated with them. We manage to adjust the behavior of our qubits and influence the outcome of the quantum operations by using the external magnetic field, which is called bias. However, in quantum computing, we don't want to control just a single qubit. Instead, we manipulate multiple qubits that are entangled with each other. To achieve this, we utilize a device known as a coupler, which essentially links the qubits together. The

purpose of the coupler is to program them to influence each other in a desired manner.

The process of quantum annealing, which takes place in a matter of microseconds, is as follows. The system starts with a set of qubits and each of them is superposition but not yet coupled. Then, the quantum annealing starts, and the values of biases of each qubit and the strength of the coupling between the qubits are adjusted. This defines the energy landscape and entangles the qubits, resulting in a system in a state of many possible answers. When the annealing is complete, what we get is each qubit in a classical state that represents the minimum energy state of the problem/landscape.

As we have seen so far, we express the problem of finding the minimum of an energy landscape in the form of a problem Hamiltonian, and we formulate an objective function in a way that it finds the solution to the problem. In quantum annealing, the system starts with an initial Hamiltonian, which has the lowest energy state when all qubits are in a superposition. The initial Hamiltonian is also called the tunneling Hamiltonian. Then, the annealing process starts, and a new Hamiltonian, the final Hamiltonian, is introduced. This Hamiltonian contains the biases and couplers, and it slowly starts to lessen the impact of the initial Hamiltonian. By the time the annealing ends, there is only a state of the final Hamiltonian. The evolution of the energy states is shown in the following figure. Note that s is an abstract parameter with $0 < s < 1$.

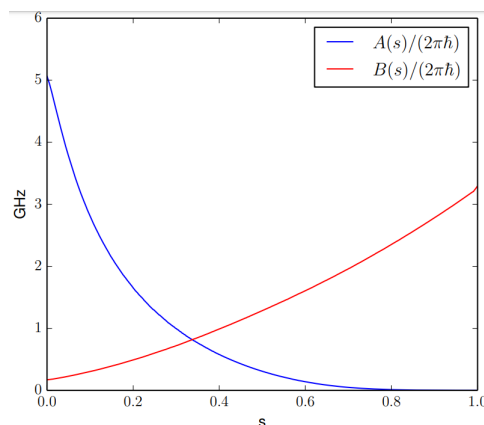


Figure 2.10: Quantum Annealing functions $A(s)$ -Initial Hamiltonian Energy- and $B(s)$ -Final Hamiltonian Energy with linear anneal.

However, for the system to stay in the minimum energy state (ground state) throughout the annealing, the process needs to be adiabatic. This means that the annealing

process should experience no interface from outside energy sources and should evolve the Hamiltonian slowly enough. By running the annealing process too fast (rapid changes) the system may deviate from the ground state. Other factors that may cause the same unwanted effect are the noise and/or errors and the temperature; higher temperatures can introduce thermal fluctuations, which cause solutions with higher energies. In reality, no computation can be performed in complete isolation, so quantum annealing can be seen as the practical version of adiabatic quantum computing.

2.5 Variational Hybrid Quantum-Classical Algorithms

Variational Quantum Algorithms (VQAs) represent a category of quantum algorithms that combine classical and quantum computing resources to find approximate solutions to problems. This approach is useful in solving optimization and eigenvalue problems with applications in considerable fields such as chemistry (simulate molecular structures), logistics (solve complex logistical problems), and finance (portfolio optimization). It is worth noting that variational quantum algorithms are typically classified as hybrid algorithms since they use a quantum-classical model of computation. Their hybrid nature and adaptability provide a versatile framework that can be used to solve a variety of problems.

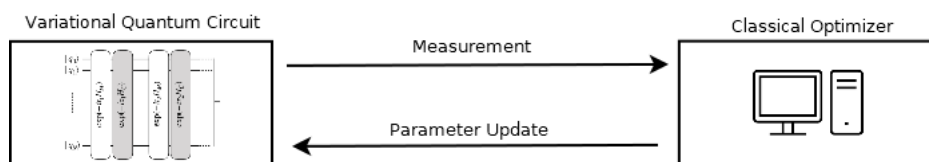


Figure 2.11: A hybrid quantum-classical computational approach with the parameterized quantum circuit running on the quantum computer and then outsourcing the parameter optimization to a classical optimizer.

Depending on the problem, the algorithm's structure and complexity vary. However, the basic elements/ building blocks of VQAs are the same. The first step is to define a cost function C , which encodes the problem's solution. Then a quantum operation -ansatz- which depends on parameters θ is created with the purpose of solving the optimization task

$$\theta^* = \arg_{\theta} \min C(\theta)$$

The main idea of the VQAs is that they use a quantum computer to estimate the cost function $C(\theta)$ and then a classical computer to estimate the parameter θ in order to minimize/maximize the energy value

$$C(\theta) = \langle \psi(\theta) | H | \psi(\theta) \rangle$$

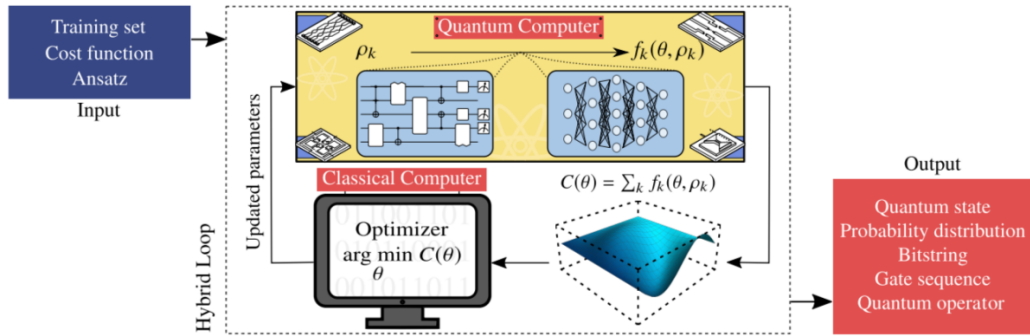


Figure 2.12: Diagram of a VQA

The general structure of a hybrid quantum-classical algorithm is shown in this workflow 2.12. The process begins with the input, which consists of a training set of data (for example, for a machine learning application), the cost function to be minimized, and the particular ansatz for the wavefunction under study. Within the quantum computer section, the state ρ_k is processed through quantum circuits by preparing the ansatz with a set of initial parameters. This setup generates the necessary functions and measures the corresponding cost function.

After the quantum measurements, the results are transferred to a classical computer where the cost function $C(\theta)$ is minimized. This minimization is done by updating the parameters θ using an optimizer. The updated parameters are then fed back into the quantum computer for further iterations. This iterative process continues in a loop until the final output is achieved.

Depending on the specific application and goal of the algorithm, the final output can be a quantum state representing the ground state of a Hamiltonian, a probability

distribution, a bitstring encoding the solution to an optimization problem, a gate sequence, or a quantum operator.

Variational Quantum Algorithms hold a lot of prospects for near-term quantum computing and can provide quantum advantages in solving diverse problems. Notably, improved ansatzes could potentially reduce the impact of noise on VQAs and improve accuracy.

Following, we will analyze the quantum variational algorithm: Quantum approximate optimization algorithm.

2.6 Quantum Approximate Optimization Algorithm (QAOA)

The Quantum Approximate Optimization Algorithm (QAOA), introduced in 2014 by Farhi, Goldstone, and Gutmann, is a hybrid quantum algorithm that was designed to solve combinatorial optimization problems by producing approximate solutions. As previously mentioned, hybrid algorithms blend classical and quantum computing by combining quantum state preparation and measurement with classical optimization. More precisely, QAOA is a variational algorithm consisting of a parameterized quantum circuit, called an ansatz, that depends on some variational parameters that are optimized classically concerning the problem. The objective is to minimize the cost function by finding an approximate best solution for the problem.

In the QAOA a circuit with p levels (depth), specified by $2p$ variational parameters for p levels/layers –so the total number of optimizable parameters is equal to $2p$ –, takes as an input a prepared quantum state. QAOA starts with the initial state $|s\rangle$ being a uniform superposition over n computational basis states, which are constructed by applying $H^{\otimes n}$.

$$|s\rangle = |+\rangle^{\otimes n} \frac{1}{\sqrt{2^n}} \sum_{b \in \{0,1\}^n} |b\rangle$$

Then, for any integer $p \geq 1$ and $2p$ angles, an angle-dependent quantum state $|\vec{\gamma}, \vec{\beta}\rangle$ is

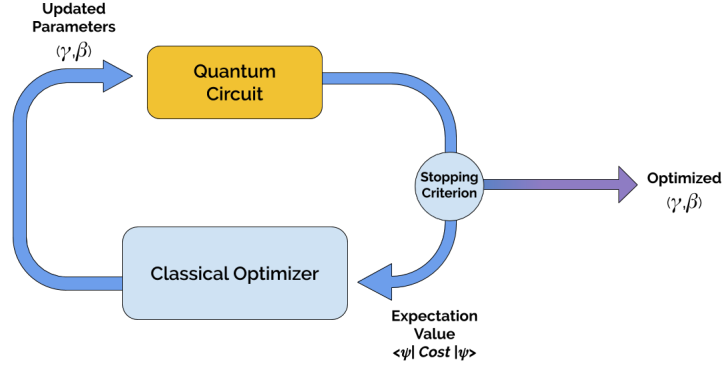


Figure 2.13: A diagram of how the variational QAOA works

obtained by applying quantum operators to the initial state

$$|\vec{\beta}, \vec{\gamma}\rangle = U(H_B, \beta_p) U(H_C, \gamma_p) \dots U(H_B, \beta_1) U(H_C, \gamma_1) |s\rangle \Leftrightarrow$$

$$|\vec{\beta}, \vec{\gamma}\rangle = \prod_{i=1}^p U(H_B, \beta_i) U(H_C, \gamma_i) |s\rangle$$

It is evident that the algorithm is based on two operators, and those quantum operators correspond to some rotations on qubits. In particular, the unitary operators $U(H_C, \gamma)$ that are applied to the quantum state and depend on an angle γ are defined as

$$U(H_C, \gamma) = e^{-i\gamma H_C} = \prod_{a=1}^m e^{-i\gamma H_{C_a}}$$

with γ lying between 0 and 2π because H_C has integer eigenvalues and with all terms commuting because they are diagonal in the computational basis.

The operator/ cost Hamiltonian H_C typically represents the problem's cost function, and it is defined in a way that its highest energy state encodes the solution to the optimization problem. For example, let's recall the MaxCut Hamiltonian

$$H = \frac{1}{2} \sum_{i,j \in E} (1 - Z_i Z_j)$$

The mixing operators $U(H_B, \beta)$, controlled by the parameter β , which follows the

unitary operator are defined as

$$U(H_B, \beta) = e^{-i\beta H_B} = \prod_{j=1}^n e^{-i\beta H_{C_j}}$$

with β lying between 0 and π .

As for the operator/ mixer Hamiltonian H_B is the sum of all single bit σ^x operators

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

with σ_j^x denoting the Pauli X-gate σ^x that acts on the j -th qubit. All the terms in the mixer Hamiltonian commute, although it is worth mentioning that the mixer Hamiltonian H_B does not commute with the cost Hamiltonian H_C . Typically, the QAOA is initialized with the mixer Hamiltonian's highest energy eigenstate.

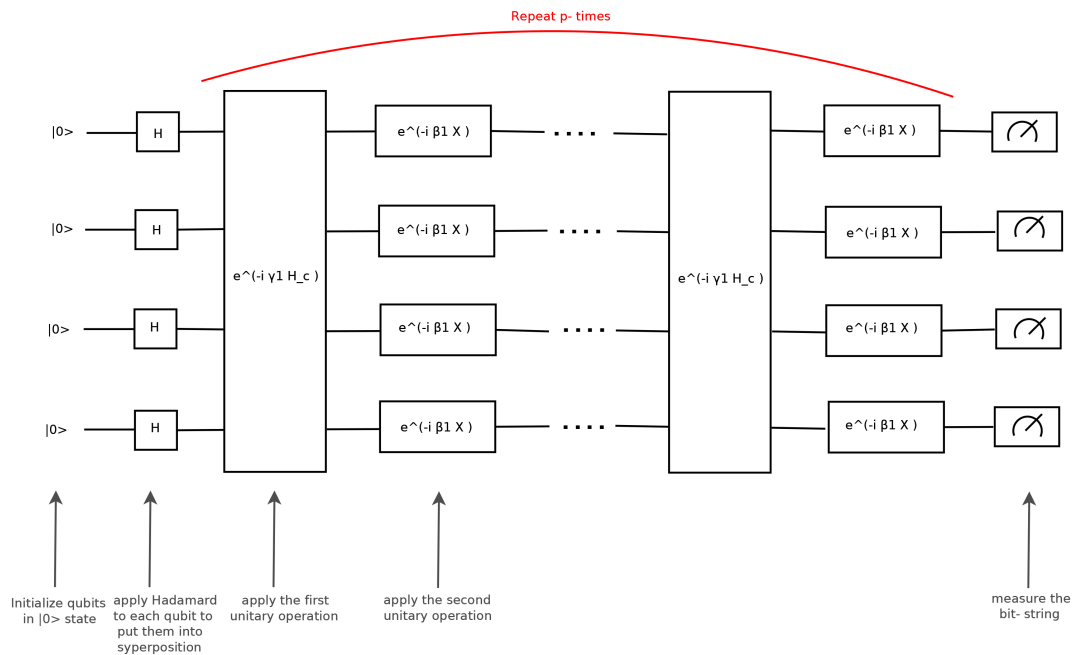


Figure 2.14: First Look of the QAOA Circuit Diagram

We call $U(H_C, \gamma)$ and $U(H_B, \beta)$ the cost and the mixer layers, respectively. The depth of the circuit p scales with the number of layers applied, with the single QAOA layer containing one cost and one mixer layer. We will look into the implementation of each layer separately.

2.6.1 Cost Layer

In the cost Hamiltonian, each of the two-qubit interactions (Z_i, Z_j) can be implemented with two CNOT quantum gates and a local one-qubit R_z gate. Let us see why. Firstly, we will take an arbitrary quantum state and for depth $p = 1$, we will apply the unitary operation

$$U(H_C, \gamma) |0 \dots 00\rangle = e^{-i\gamma H_C} |0 \dots 00\rangle$$

We observe that to implement the operation, we need to exponentiate the Z-Pauli gate. In general, the exponentiation of the Pauli gates corresponds to rotation gates. For a matrix M , the matrix exponential is defined as the power series:

$$e^M = \sum_{k=0}^{\infty} \frac{1}{k!} M^k$$

For the X-Pauli matrix $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ the exponentiation is a rotational X gate with a rotation angle (here we denote the angle with 2θ):

$$e^{-i\theta X} = R_X(2\theta) = \begin{bmatrix} \cos(\theta) & -i\sin(\theta) \\ -i\sin(\theta) & \cos(\theta) \end{bmatrix}$$

Notice that in order to rotate around z -axis with an angle θ , we give as an input to R_X an angle 2θ . This happens because the Bloch sphere uses a half-angle representation for representing any state on the sphere.

Analogously for the Y-Pauli matrix $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ we have the rotation angle

$$e^{-i\theta Y} = R_Y(2\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

And for the Z-Pauli matrix $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ we get the rotation angle

$$e^{-i\theta Z} = R_Z(2\theta) = \begin{bmatrix} e^{-i\theta} & 0 \\ 0 & e^{-i\theta} \end{bmatrix}$$

So, it becomes clear that the exponentiation of the cost Hamiltonian turns all the single Z -gates into rotational Z -gates. The result is called ansatz Z -gate, and it is a control single-qubit rotational gate that is applied to each single qubit. So, an ansatz Z -gate is applied to a qubit pair for each interaction term in the cost Hamiltonian. Nevertheless, the cost Hamiltonian is a sum of matrices that commute, so we split the sum into a product of exponential terms, and we have:

$$Z \otimes Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So, the exponentiation of this diagonal matrix looks like

$$e^{-i\theta Z \otimes Z} = \begin{bmatrix} e^{-i\theta} & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & e^{-i\theta} \end{bmatrix}$$

This quantum gate is called R_{zz} and is the gate that is used -the one that can be decomposed into two CNOT and one R_z gate. So, the notation will be

$$R_{ZZ}(2\Theta) = e^{-i\theta Z \otimes Z} = \begin{bmatrix} e^{-i\theta} & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & e^{-i\theta} \end{bmatrix}$$

It is easy to recognize that this is a direct sum of R_Z rotations, and so we can write the equivalent to this gate, which is a uniformly controlled (multiplexed) R_Z gate

$$R_{ZZ}(2\Theta) = \begin{bmatrix} R_Z(2\theta) & 0 \\ 0 & R_Z(2\theta) \end{bmatrix}$$

But for now, let's see how we can simulate the operator corresponding to the problem Hamiltonian without examining the matrices. We will replace in the unitary operation, the cost Hamiltonian H_C with the objective function $\frac{1}{2} \sum_{i,j \in E} (1 - Z_i Z_j)$, as mentioned, and we have

$$U(H_C, \gamma) |0 \dots 00\rangle = e^{-i\gamma \frac{1}{2} \sum_{i,j \in E} (1 - Z_i Z_j)} |0 \dots 00\rangle \Leftrightarrow$$

$$U(H_C, \gamma) |0 \dots 00\rangle = \prod_{i,j \in E} e^{-i\gamma \frac{1}{2} (1 - Z_i Z_j)} |0 \dots 00\rangle$$

We analyze a term of the product, and we write that

$$e^{-i\gamma \frac{1}{2} (1 - Z_i Z_j)} |0 \dots 00\rangle = e^{-i\gamma \frac{1}{2}} e^{+i\gamma \frac{1}{2} Z_i Z_j} |0 \dots 00\rangle$$

We will take into account a theorem of linear algebra that states:

Theorem 2.6.1. $e^A |u\rangle = e^\lambda |u\rangle$, if and only if $A |u\rangle = \lambda |u\rangle$

And by keeping in mind that the Z 's eigenvectors are $|0\rangle, |1\rangle$ and its eigenvalues are ± 1 , the operator $e^{-iZZt} = e^{-iZ \otimes Zt}$ have four different possibilities:

$$e^{-iZ \otimes Zt} |00\rangle = e^{-i(1 \times 1)t} |00\rangle = e^{-it} |00\rangle$$

$$e^{-iZ \otimes Zt} |01\rangle = e^{-i(1 \times -1)t} |01\rangle = e^{it} |01\rangle$$

$$e^{-iZ \otimes Zt} |10\rangle = e^{-i(-1 \times 1)t} |10\rangle = e^{it} |10\rangle$$

$$e^{-iZ \otimes Zt} |11\rangle = e^{-i(-1 \times -1)t} |11\rangle = e^{-it} |11\rangle$$

We notice that a phase factor with the sign depending on parity is added. So, in general:

$$e^{-iZ \otimes Z t} = e^{-i(-1)^{a \oplus b} t} |ab\rangle$$

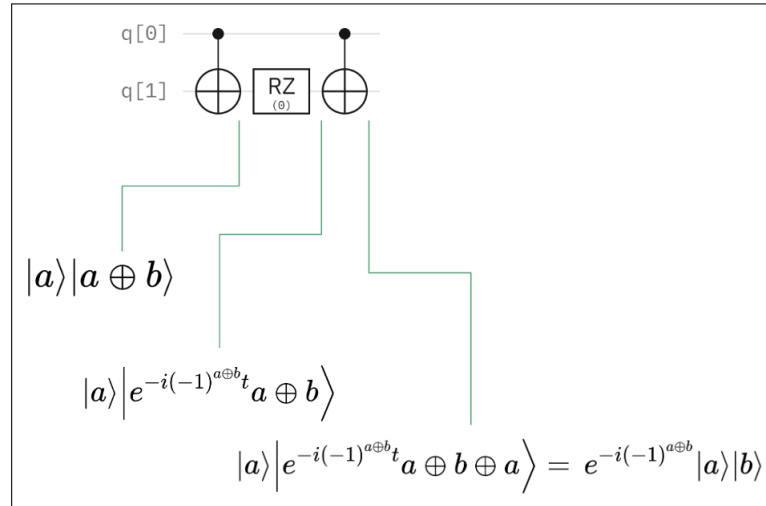


Figure 2.15: Implementation of cost Hamiltonian; shows how 2 CNOT and 1 **R_Z** gate is equivalent with an **R_{ZZ}** gate

In the case the Hamiltonians commute (such as in the cost layer), we can just concatenate corresponding circuits:

$$e^{-i\alpha Z_i Z_j t - i b Z_j Z_k t} = e^{-i\alpha Z_i Z_j t} e^{-i b Z_j Z_k t}$$

and so we have

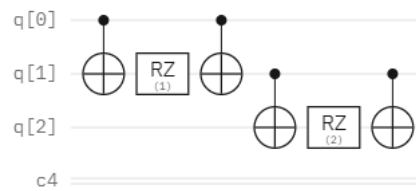


Figure 2.16: Quantum circuit for the unitary operation

2.6.2 Mixer Layer

As previously stated, the mixer Hamiltonian is defined as the sum of all Pauli-X operators.

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

So, the unitary operator can be analyzed as

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

Like the cost layer, we take the exponential of a Pauli gate, and so we get the respective rotational gate. Here, all the single X-gates turn into rotational X-gates.

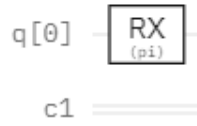


Figure 2.17: Implementation of mixer Hamiltonian

Before we finish with the presentation of the QAOA layers, we should also mention the correlation between the two diagonalized operations used in those two layers. We take the Taylor series

$$e^M = \sum_{k=0}^{\infty} \frac{1}{k!} M^k$$

for the exponent e^{-iZt} and we diagonalized it by “sandwiching” in between Hadamards gates

$$e^{-iZt} = \sum_{j=0}^{\infty} \frac{(-iZt)^j}{j!} = I - iZt - \frac{Z^2 t^2}{2!} + \dots$$

We have that

$$\begin{aligned} H e^{-iZt} H &= H \left(I - iZt - \frac{Z^2 t^2}{2!} + \dots \right) H = I - iH Z H t - \frac{H Z H H Z H t^2}{2!} + \dots = \\ &= I - iXt - \frac{X^2 t^2}{2!} = e^{-iXt} \end{aligned}$$



Figure 2.18: Implementation of gate

With all of the above, we now know how to simulate the entire QAOA circuit. Let us go back one step and remember that the final state output is given by the circuit after the initialization of the $2p$ parameters $\vec{\gamma} = (\gamma_1, \gamma_2 \dots \gamma_p)$, $\vec{\beta} = (\beta_1, \beta_2 \dots \beta_p)$ and the definition of the total number of layers p is

$$|\vec{\beta}, \vec{\gamma}\rangle = U(H_B, \beta_p) U(H_C, \gamma_p) \dots U(H_B, \beta_1) U(H_C, \gamma_1) |s\rangle$$

$$|\vec{\beta}, \vec{\gamma}\rangle = e^{-i\beta_p H_B} e^{-i\gamma_p H_C} \dots e^{-i\beta_1 H_B} e^{-i\gamma_1 H_C} |s\rangle$$

where $|s\rangle$ is the equal superposition of the 2^n states.

For each layer p the algorithm applies the unitary $U(H_C, \gamma_p)$ followed by the mixing unitary $U(H_B, \beta_p)$. After receiving the final state, we repeat measurements of the output on a computational basis to get the cost obtained by the algorithm for a specific problem. We define the expectation value of the cost Hamiltonian H_C as

$$F_p(\vec{\beta}, \vec{\gamma}) = \langle \psi(\vec{\beta}, \vec{\gamma}) | H_C | \psi(\vec{\beta}, \vec{\gamma}) \rangle$$

The objective is to find the optimal set of parameters $(\vec{\beta}, \vec{\gamma})$ which maximizes the expectation value F_p . This is achieved through classical optimization, wherein the angles are iteratively updated. So we denote the maximum of F_p over the angles as

$$M_p = \max_{\vec{\beta}, \vec{\gamma}} F_p(\vec{\beta}, \vec{\gamma}).$$

At the end of the optimization procedure, the state $|\psi(\vec{\beta}, \vec{\gamma})\rangle$ will encode the solution to the underlying optimization problem.

It's worth noting that due to its design, QAOA shares a close connection with Adiabatic and so Quantum Annealing. The algorithm's performance is theoretically expected to improve as the number of layers, denoted as p , increases, and so the algorithm yields a performance guarantee for limit $p \rightarrow \infty$

$$\lim_{p \rightarrow \infty} M_p = C_{\max}$$

It is easy to understand that this also means that the value for a deeper QAOA circuit (M_{p+1}) is always at least as good as the value of a less deep QAOA circuit (M_p)

$$M_{p+1} \geq M_p$$

We consider the time-dependent Hamiltonian $H(t) = \frac{t}{T}H_C + (1 - \frac{t}{T})H_M$ which evolves out a system where H_C is the cost Hamiltonian and H_M is the mixed Hamiltonian (sum of all X operators). Note that Pauli X-gate has eigenvalues $+1, -1$, and hence that initial QAOA state is the highest energy state of H_M .

2.6.3 Solving MaxCut Problem with Quantum Approximate Optimization Algorithm

Now, we can solve the MaxCut problem for different graphs. We will start with creating a 5-nodes fully connected graph (G). The first step is to construct the graph.

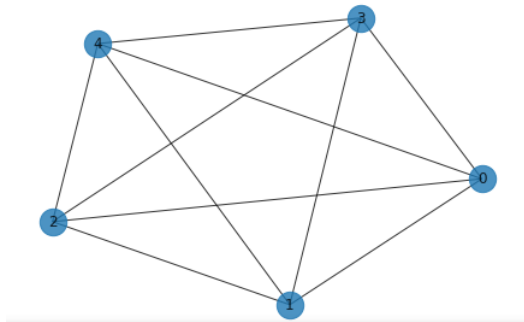


Figure 2.19: A 5-nodes fully connected graph

Then we will construct the mixer layer (figure 2.20) which consists of R_X gates and is parameterized by β (beta) parameter. We then continue by constructing the cost

layer (figure 2.21), which is parameterized by γ (gamma) parameter. Finally, let's not forget that we also have to create the initial state, which is to apply the Hadamard gate to every qubit to cause a superposition state. Since we have prepared the three elements that compose the QAOA circuit, we are now able to see the full quantum circuit for QAOA, which was created for this problem (figure 2.22). Note that this is a fully connected graph problem, so we have the mixer operations applied between every possible qubit pair.

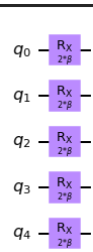


Figure 2.20: Mixer layer- apply on each qubit the R_x gate parameterized by 2β



Figure 2.21: Cost layer- apply 2 CNOT gates and an R_z gate (decomposition of R_{zz} gate)

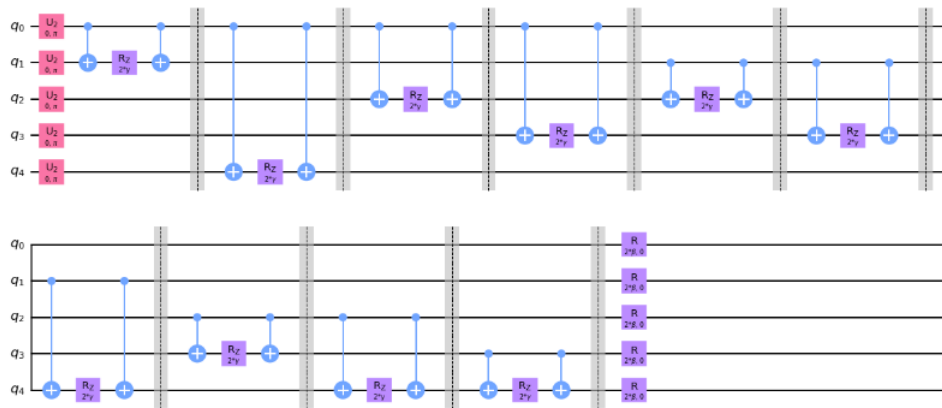


Figure 2.22: QAOA circuit - Between each barrier (grey dotted lines), there are the mixer operators $U(H_c, \gamma)$ for each edge

Now we have to find the optimal parameters β , γ (beta and gamma) that minimizes the expectation value $\langle \vec{\beta}, \vec{\gamma} | H | \vec{\beta}, \vec{\gamma} \rangle$. The process that is followed is; we start with the function that returns the number of edges shared between the two partitions of our graph,

then based on the measurement results from the previous function, we compute the expectation value (the mean of cut values), and after that, we create the parameterized QAOA circuit by bringing together all the different circuit components that we built before. The inputs of the function are the graph G and the angle θ (theta) and we are able to compute the number of repetitions for the unitaries. Next, we take the output of the previous function, which is the qiskit parameterized circuit, and we run it. The inputs of this function are the graph G , the number of repetitions of the unitaries p and the number of iterations of running the circuit, shots. For this example, we choose COBYLA (Constrained Optimization BY Linear Approximation) as our classical optimization algorithm.

```

:      fun: -5.91015625
      maxcv: 0.0
      message: 'Optimization terminated successfully.'
      nfev: 33
      status: 1
      success: True
      x: array([4.43997891, 0.20411664])

```

Figure 2.23: After 33 evaluations, we get $M_{p=1}(\gamma, \beta) = 5.91$

Notice that the results of the optimizer are negative because MaxCut is a maximization problem, but Python can only save minimization problems, hence the minus. Now, we present the histogram of all the possible solutions to the problem (2.24). Notice how the outputs have somewhat high probabilities.

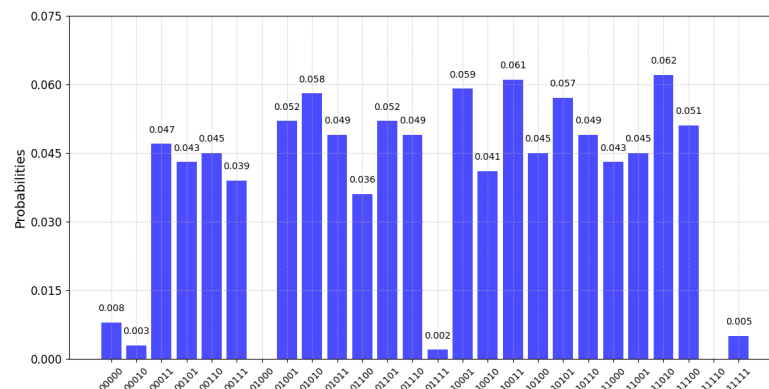


Figure 2.24: Distribution of states for the fully connected graph with optimal parameters

Given Figure 2.24, the maximum cut here is the cut for the most frequent state (11010).

We will now construct a 15-node fully connected graph, and we will focus on finding the optimal angles by solving the graph with QAOA again. We present the graph in figure 2.25. First, we get the results for $p = 1$ and calculate the cost of MaxCut from the probabilities of the final state with $M_1 = 7.607401005446704$. In figure 2.26, we present a diagram of how the cost function evolves through iterations for the problem.

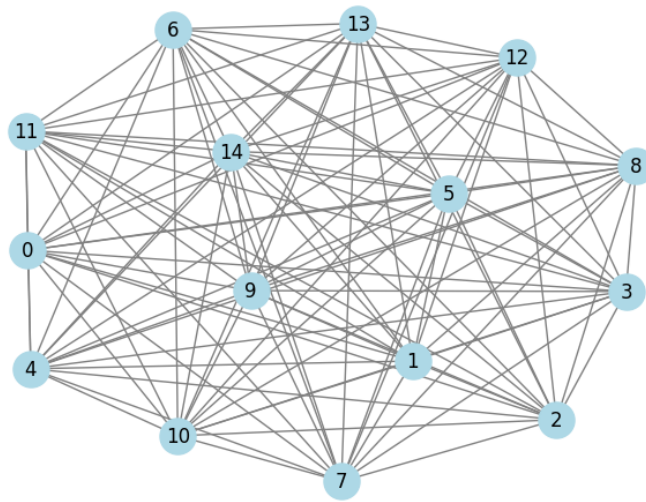


Figure 2.25: 17-nodes fully connected graph

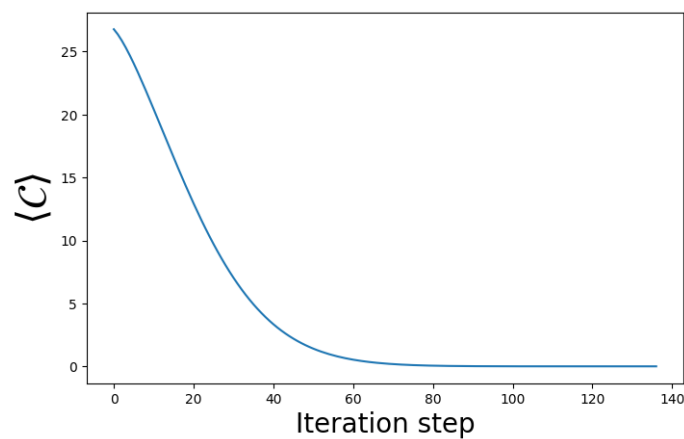


Figure 2.26: Plot of the cost functions of the QAOA over iteration steps for $p = 1$.

To find the optimal angles, we start intentionally from a bad point, and we get the best angles to be $\gamma_1 = 0.5419451229548377$ and $\beta_2 = -0.0002848347092071101$. In

figure 2.27, we present a heatmap of the costs of the QAOA for different values of the parameters γ , β using [30]. Note that we also map the best solution to the problem with a pink-x (Global optimum). We continued the process, but this time, we set $p = 2$ and updated the angles for the optimizer. Then, we reran the quantum circuit. In figure 2.28, we present the cost function for each iteration.

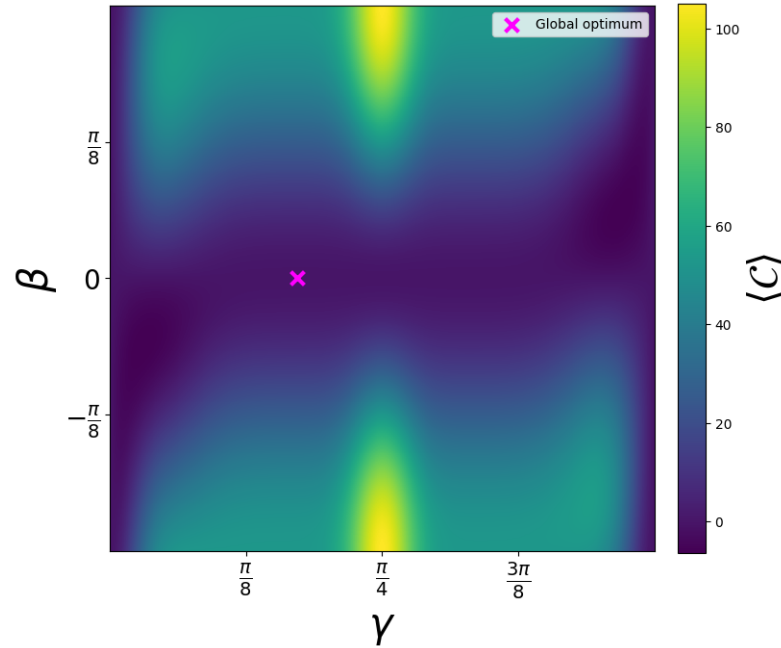


Figure 2.27: A heatmap for different values of parameters γ , β . Note how the better solutions are represented by darker colors (lower cost values) and the worst solutions are represented by brighter colors (higher cost values).

Iteration	6	Cost =	-0.0292
Iteration	14	Cost =	-0.2486
Iteration	21	Cost =	-1.2644
Iteration	27	Cost =	-3.3493
Iteration	34	Cost =	-5.8192
Iteration	42	Cost =	-6.4837
Iteration	49	Cost =	-6.5543
Iteration	56	Cost =	-6.7366
Iteration	62	Cost =	-6.8570
Iteration	70	Cost =	-6.8767
Iteration	78	Cost =	-6.8886

Figure 2.28: Iteration/Cost function pairs

Now, for the updated circuit with layer $p = 2$, we plot the cost function for each iteration again, and we notice that the diagram is slightly changed. This change could be caused by the increased complexity of the quantum state, which can result in a more complex landscape of the cost function. For the value of p being equal to 2, the preparation of

more complex states becomes possible, and this is due to the fact that the circuit in use is deeper than before.

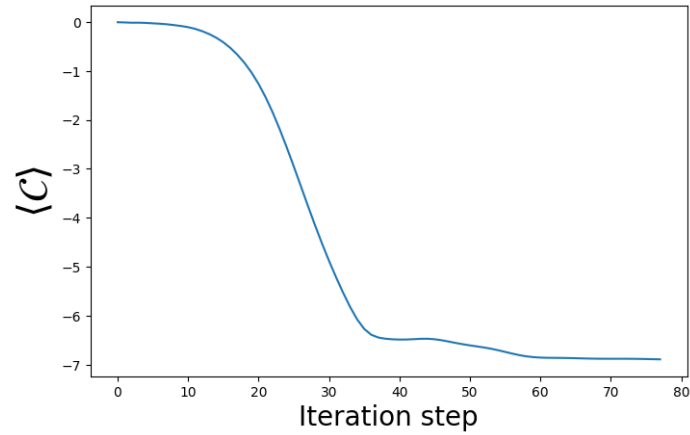


Figure 2.29: Plot of the cost functions of the QAOA over iteration steps for $p = 2$.

The optimal angles that we get are $\gamma_2 = [0.07667681 \ 0.30964072]$ and $\beta_2 = [-0.21560314 \ -0.05411769]$.

Chapter 3

Quantum Optimization for Scheduling Problems

In the upcoming sections of this chapter, we will explore the application of quantum optimization approaches to scheduling problems. We will start with an introduction to the concept of scheduling problems and their industrial applications. Then, we will focus on the Nurse Scheduling Problem, presenting its formulation and the results of solving scheduling problems using fixed inputs with varying levels of complexity and topologies in a quantum annealing device. From simple to complex problem instances and different topologies, we aim to evaluate the performance and scalability of the solutions.

3.1 Scheduling Problems

Scheduling is a decision-making process used regularly in many manufacturing and services industries. Some examples of such problems are gates assignments at airports, how to allocate runways in an airport to deal with takeoffs and landings, in computing, how to allocate CPU to programs that need to be executed, in hospitals, how to allocate the staff, medical doctors to the patients that may need different services, shift organizing. Scheduling problems pertain to establishing both the timing and use of resources within an organization.

In a scheduling problem, there is a set of jobs for completion. Each of these jobs

can be divided into a series of tasks that must be completed in a certain order, and to complete these tasks -and so the jobs- a set of resources exists. The scheduling problems are trying to provide the best answer on how to allocate each of these tasks to the different available resources. The goal is to allocate jobs to resources over given time periods to optimize some measure performance, ex. the length of time.

According to Nagar, Haddock, and Heragu (1995), scheduling problems are classified based on several criteria: the nature of the problem, the number of machines involved, the method of solving the problem, and the performance measures of the schedule. The scheduling process is divided into two classes of problems: deterministic scheduling and uncertain scheduling, based on the nature of the problem (Figure 3.1).

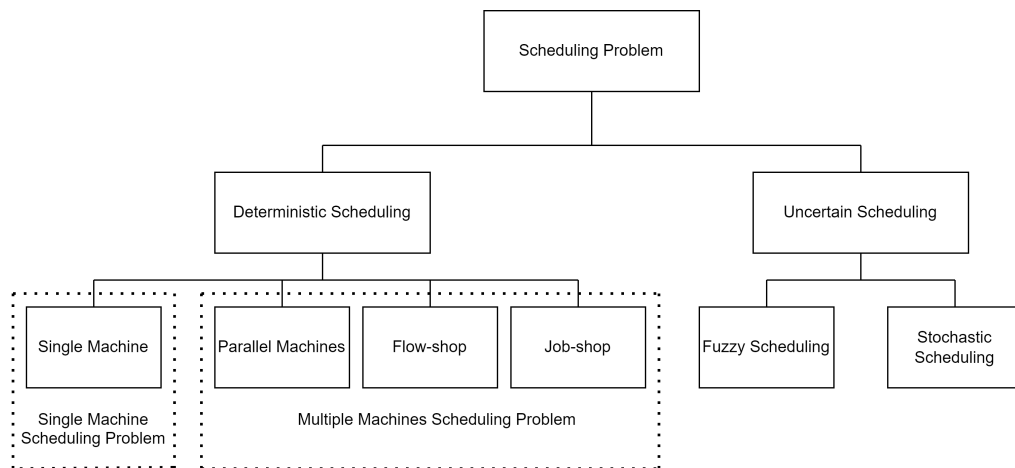
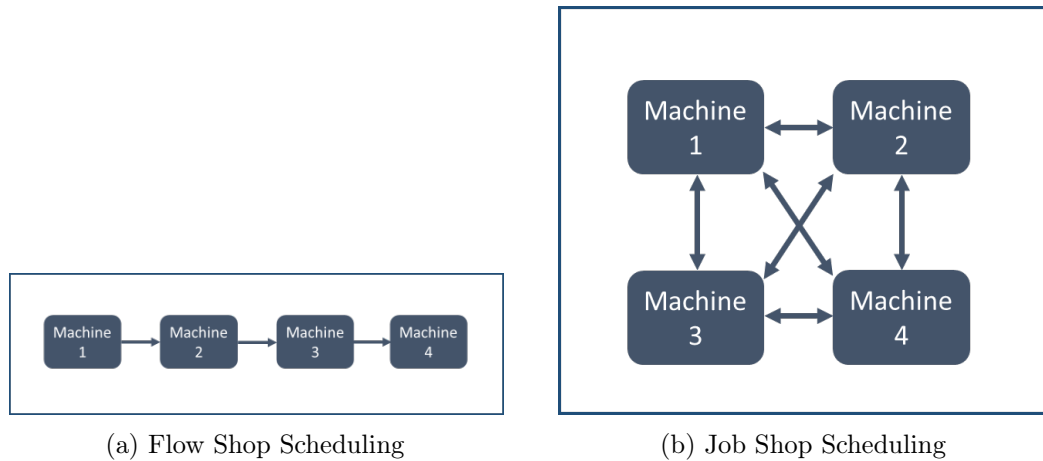


Figure 3.1: The classification of scheduling problems according to Nagar, Haddock & Heragu, 1995

Here, we examine the deterministic multiple machines scheduling problems (Figure 3.2). In parallel machine scheduling, the jobs to be scheduled are processed on a series of same-function machines. In a flow shop, all orders follow a fixed linear structure, meaning that the orders need to be manufactured in the same way on the same machines. This process could create a more rigid production flow. In contrast, a job shop, allows for individual routing of each job. This means that all orders can potentially be manufactured differently on the same machines or a specific part of the same machines, providing more flexibility in the production process. For a detailed problem definition and two different quantum formulations of the JSSP, please refer to Appendix A.



(a) Flow Shop Scheduling

(b) Job Shop Scheduling

Figure 3.2: Types of Scheduling Problems.

Source

Within this broad landscape of scheduling problems, the nurse scheduling problem (NSP) emerges as a specialized case. The NSP integrates the fundamental principles of scheduling with additional constraints unique to the healthcare industry, such as legal regulations, shift patterns, and individual preferences, highlighting the progression and interconnection from general scheduling frameworks to specific, real-world applications.

3.2 Nurse Scheduling Problem (NSP)

Let's take a closer look at the Nurse Scheduling Problem (NSP), a significant scheduling problem. This chapter will demonstrate the conversion of the NSP into a QUBO problem using the methods we have previously discussed. Afterward, we will utilize a quantum annealer provided by D-Wave on the cloud to find solutions for the NSP.

The Nurse Scheduling Problem is well-studied in mathematical optimization and is of known complexity, NP-hard. The goal is to find the optimal schedule for a set of available nurses over a fixed timetable of shifts. Depending on the specific problem at hand, various constraints must be taken into account, and solutions are required to respect them, with certain constraints being of greater significance than others. For example, the scheduling of the nurses' days off or ensuring their minimum availability are considered hard constraints. In contrast, constraints related to minimum shift assignments are categorized as soft constraints.

Undoubtedly, the Nurse Scheduling problem is a very common problem every hospital faces every day since it is nothing more than a plan for all nurses and how to create a satisfying schedule that assigns nurses to shifts or shift categories by respecting some constraints.

3.2.1 Problem Formulation

Let us now dive into the formulation of the Nurse Scheduling problem.

To construct the objective function, we consider a set of N nurses with $n = 1, \dots, N$ and a schedule consisting of D working days with $d = 1, \dots, D$. We also use the binary variable $q_{n,d} \in \{0, 1\}$ to indicate whether nurse n is assigned to day d , and so we have

$$x_{i,t} = \begin{cases} 1 & \text{if nurse } n \text{ is assigned to day } d \\ 0 & \text{otherwise} \end{cases}$$

Furthermore we define composite indices $i(n, d)$ and $j(n, d)$ as the functions of the day d and the nurse n .

As mentioned before, the scheduling problem of optimizing the assignment of nurses to working days involves considering several constraints. The objective function must correspond to these constraints. Each term in the objective function penalizes different aspects of the problem, including hard shift constraints, hard nurse constraints, and soft nurse constraints. Let's break down each term of the objective function.

First, we will construct the hard nurse constraint, which states that a nurse n must not work two or more consecutive working days. We want to encourage the schedule to avoid the assignment of nurses to work two consecutive days. To penalize the violation of the constraint, we introduce the symmetric, real-valued matrix J . The matrix J contains the penalty value a -a positive correlation constant- for assigning a nurse to work on two consecutive days such that

$$J_{i(n,d),j(n',d')} = \begin{cases} a & \text{if } n' = n \text{ and } d' = d + 1 \\ 0 & \text{otherwise} \end{cases}$$

We want to consider all the possible combinations of nurse assignments and the total penalty incurred based on the penalty values to be specified in the matrix J . Thus, we have the term

$$\sum_{n,n'}^N \sum_{d,d'}^D J_{i(n,d),j(n',d')} q_{i(n,d),j(n',d')}$$

After that, we define the hard shift constraint, which states that the required workforce on each day d must be satisfied. This constraint ensures that at least one nurse is working every day. In this term, we want to encourage the schedule to assign nurses in such a way that the total effort available on each day matches the required workforce. So, we introduce a quadratic function to penalize schedules that have either too many or too few nurses assigned. To express this term, we introduce the required workforce needed on each day d , which is denoted as $W(d)$ and the level of effort available from each nurse n , which is denoted as $E(n)$. We assume that those two functions are integer-valued functions of n or d , although this is not mandatory. We have

$$\lambda \sum_d^D \left(\sum_n^N E(n) q_{i(n,d)} - W(d) \right)^2$$

Note that λ is a positive real-valued number that tunes the significance of the term relative to other terms in the objective function by adjusting its value.

Finally, we introduce the soft nurse constraint, which states that the desired number of work days for each nurse n should be ensured. This term encourages the schedule to assign nurses in such a way that each nurse's work aligns with their desired number of work days. So, we will apply a quadratic penalty if the work schedule does not account for nurse preferences in the work schedule. With $F(n)$, we denote the number of work days that each nurse wished to be scheduled, and we stipulate that for all the nurses n , the minimum duty days $F(n)$ are

$$F(n) \geq \left\lceil \frac{D}{N} \right\rceil$$

where $\lceil x \rceil$ represents the integer part of x and $x \in \mathbb{R}$.

With $G(n, d)$ we denote the combined preference for nurse n to work on the day d . We have

$$\gamma \sum_n^N \left(\sum_d^D G(n, d) q_{i(n,d)} - F(n) \right)^2$$

Note that the total preference function can be decomposed into a product of two factors that allow for customization based on specific preferences and constraints, like

$$G(n, d) = h_1(n)h_2(n)$$

In such a way that $h_1(n)$ provide the preference level of nurse n based on their workload

$$h_1(n) = \begin{cases} 3 : & \text{busy} \\ 2 : & \text{moderate} \\ 1 : & \text{idle} \end{cases}$$

So $h_1(n) = 3$ indicates that the nurse prefers to be on a busy shift, $h_1(n) = 2$ indicates a moderate preference and $h_1(n) = 1$ indicates that the nurse prefers to be in an idler shift. The factor $h_2(d)$ represents the preference level based on the specific day d of the week

$$h_2(d) = \begin{cases} 2 : & \text{weekend} \\ 1 : & \text{weekday} \end{cases}$$

So by tuning the value $h_2(d)$, we get the preferences of nurses regarding the days they work, more specifically, whether nurses may work on weekends or weekdays. It is worth pointing out that the formulation of preference can be more sophisticated by including more preference options, like a three-shift system, a differentiation between weekdays and weekends regarding workload, or a day-off request with priority. With that being said, we could rewrite the objective function for the soft nurse constraint as

$$\gamma \sum_n^N \left(\sum_d^D h_1(n)h_2(n)q_{i(n,d)} - F(n) \right)^2$$

The positive real-value number γ allows tuning the significance of this quadratic penalty function relative to the other terms in the objective function. Adjusting γ allows balancing the importance of meeting nurse preferences against other objectives. Usually, the coefficient γ has lower values relative to the other coefficients, such as λ , because the soft nurse constraint is considered less important.

3.2.2 Objective Function Construction

Now that we have discussed the individual terms, we are ready to compose those terms into a single objective function that yields the QUBO form

$$\begin{aligned} H(q) &= \sum_{n,n'}^N \sum_{d,d'}^D J_{i(n,d),j(n',d')} q_{i(n,d),j(n',d')} \\ &+ \lambda \sum_d^D \left(\sum_n^N E(n)q_{i(n,d)} - W(d) \right)^2 \\ &+ \gamma \sum_n^N \left(\sum_d^D h_1(n)h_2(n)q_{i(n,d)} - F(n) \right)^2 \end{aligned}$$

To gain a better understanding of the Nurse Scheduling Problem, let us consider a simple example. Suppose that we have three nurses and a schedule that consists of two working days, so we have $n = 3$ and $d = 2$. The final optimal schedule that we get assigns the nurse_1 to work the first day and the nurse_2 to the second day while nurse_3 doesn't work at all (Figure 3.3).

So we have

$$\begin{cases} q_{i(n_1,d_1)} = 1 \\ q_{i(n_2,d_1)} = 0 \\ q_{i(n_3,d_1)} = 0 \end{cases} \quad \begin{cases} q_{i(n_1,d_2)} = 0 \\ q_{i(n_2,d_2)} = 1 \\ q_{i(n_3,d_2)} = 0 \end{cases}$$

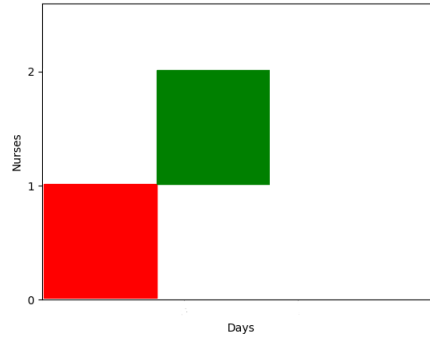


Figure 3.3: Schedule of the example

First, we will show how matrix J is constructed for our example. Then, we will demonstrate that all the conditions have been met, so there are no constraints violations. Thus, we can confidently conclude that the total objective function is minimal.

The hard nurse constraint uses the matrix J , which represents the coefficients of the quadratic terms and contains the penalty value 'a' assigned for scheduling a nurse to work on two consecutive days. The matrix is considered a dictionary with keys in the form of tuples (i, j) , where i and j are indices that represent a specific nurse on a particular day. The form of the matrix is as follows:

$$J_{i,j} = \begin{bmatrix} 0 & a & 0 & \cdots & 0 \\ a & 0 & a & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

As we mentioned before, J is symmetric, which means that for any two variables i and j , the coefficient of the quadratic term i,j is the same as the coefficient of the term j, i .

$$J_{i(n,d),j(n',d')} = J_{j(n',d'),i(n,d)}$$

This symmetry simplifies the problem by reducing the number of coefficients we need to calculate. Additionally, the values in matrix J are carefully chosen to ensure that the assigned nurses work efficiently and do not get exhausted, but for the sake of simplicity, we, for now, set the penalty value equal to one. Let's take a look at the matrix J we

obtained by solving our problem with a quantum annealer.

```
[0, 1.0, 0, 0, 0, 0, 0, 0, 0]
[1.0, 0, 1.0, 0, 0, 0, 0, 0, 0]
[0, 1.0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1.0, 0, 0, 0, 0]
[0, 0, 0, 1.0, 0, 1.0, 0, 0, 0]
[0, 0, 0, 0, 1.0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1.0, 0]
[0, 0, 0, 0, 0, 0, 1.0, 0, 1.0]
[0, 0, 0, 0, 0, 0, 0, 1.0, 0]
```

Figure 3.4: A representation of matrix J when solving the nurse scheduling problem for $N=3$ and $D=2$, with penalty $a=1.0$

The hard shift constraint ensures that the right amount of nurses is assigned to the schedule, not too many or too few. So we want the effort that is put in by the nurses to be at least as the needed workforce but not much more extensive. We could express that as

$$\textit{Effort of nurses assigned} > \textit{workforce needed}$$

or more accurate

$$\textit{Effort of nurses assigned} \cong \textit{workforce needed} \Rightarrow$$

$$E(n)q_{i(n,d)} \cong W(d)$$

Since the form of QUBO contains no constraints besides the binary variables, we reformulate our model with quadratic penalties into the objective function. The penalties are designed in a way that they become equal to zero for the feasible solutions and equal to a positive number for the infeasible solutions. These penalties are then added to the objective function. In that way, if the solution is possible, the penalty will not added to the objective function.

In our process, we make use of the QUBO model. This model allows us to accommodate both quadratic and linear functions, with a diagonal matrix being employed for the linear case. In that way, we express the constraints with an inequality that can

represent the slack variables by a binary expansion. With that process, we managed to convert a constrained quadratic optimization problem into an equivalent unconstrained QUBO model by expressing the constraints as quadratic penalties that are added to the objective function.

Here, to penalize the constraint violation, we introduce the penalty λ , a large enough positive number

$$\lambda \sum_d^D \left(\sum_n^N E(n) q_{i(n,d)} - W(d) \right)^2$$

We will prove that this penalty term is equal to zero when all the constraints are satisfied.

Assume that $W(d) = E(n) = 1$.

$$\begin{aligned} & \lambda \sum_d^2 \left(\sum_n^2 E(n) q_{i(n,d)} - W(d) \right)^2 \xrightarrow{E(n)=1} \\ & \lambda \sum_d^2 \left(\sum_n^2 q_{i(n,d)} - W(d) \right)^2 = \\ & \lambda \sum_d^2 \left(q_{i(n_1,d)} + q_{i(n_2,d)} + q_{i(n_3,d)} - W(d) \right)^2 = \\ & \lambda \sum_d^2 \left(q_{i(n_1,d)}^2 + q_{i(n_2,d)}^2 + q_{i(n_3,d)}^2 + 2q_{i(n_1,d)}q_{i(n_2,d)} \right. \\ & \quad \left. + 2q_{i(n_2,d)}q_{i(n_3,d)} + 2q_{i(n_1,d)}q_{i(n_3,d)} - 2W(d)q_{i(n_1,d)} \right. \\ & \quad \left. - W(d)q_{i(n_2,d)} - 2W(d)q_{i(n_3,d)} + W(d)^2 \right) \xrightarrow{W(d)=1} \end{aligned}$$

$$\begin{aligned} & \lambda \left(q_{i(n_1,d_1)}^2 + q_{i(n_2,d_1)}^2 + q_{i(n_3,d_1)}^2 + 2q_{i(n_1,d_1)}q_{i(n_2,d_1)} \right. \\ & \quad \left. + 2q_{i(n_2,d_1)}q_{i(n_3,d_1)} + 2q_{i(n_1,d_1)}q_{i(n_3,d_1)} - 2q_{i(n_1,d_1)} \right. \\ & \quad \left. - 2q_{i(n_2,d_1)} - 2q_{i(n_3,d_1)} + 1^2 + q_{i(n_1,d_2)}^2 + q_{i(n_2,d_2)}^2 \right. \\ & \quad \left. + q_{i(n_3,d_2)}^2 + 2q_{i(n_1,d_2)}q_{i(n_2,d_2)} + 2q_{i(n_2,d_2)}q_{i(n_3,d_2)} \right. \\ & \quad \left. + 2q_{i(n_1,d_2)}q_{i(n_3,d_2)} \right) \end{aligned}$$

$$\begin{aligned}
& - 2q_{i(n_1, d_2)} - 2q_{i(n_2, d_2)} - 2q_{i(n_3, d_2)} + 1^2 \Big) \\
& q_{i(n_2, 3, d_1)} = 0 \\
& \underline{\underline{q_{i(n_1, 3, d_2)} = 0}} \\
& \lambda \left(q_{i(n_1, d_1)}^2 + 2q_{i(n_1, d_1)} \cdot 0 + 2q_{i(n_1, d_1)} \cdot 0 \right. \\
& - 2q_{i(n_1, d_1)} - 2 \cdot 0 - 2 \cdot 0 + 1^2 + q_{i(n_2, d_2)}^2 \\
& + 2q_{i(n_2, d_2)} \cdot 0 + 2q_{i(n_2, d_2)} \cdot 0 - 2 \cdot 0 \\
& \left. - 2q_{i(n_2, d_2)} - 2 \cdot 0 + 1^2 \right) \\
& q_{i(n_1, d_1)} = 1 \\
& \underline{\underline{q_{i(n_2, d_2)} = 1}} \\
& \lambda \left(1^2 - 2 + 1^2 + 1^2 - 2 + 1^2 \right) = 0
\end{aligned}$$

We proved that the quadratic penalty does become equal to zero when the constraints are satisfied, and so the objective function becomes minimum.

We will repeat the same process for the soft nurse constraint, too. The soft nurse constraint ensures that the preference of nurse n to work on the day d aligns with the general duty days of the hospital. So we want the preference of nurses to match the number of work days that each nurse has to be scheduled

$$h_1(n)h_2(n)q_{i(n, d)} > F(n)$$

$$h_1(n)h_2(n)q_{i(n, d)} - F(n)$$

Or if we want to be more specific

$$\textit{Preference} \cong \textit{Duty days}$$

$$h_1(n)h_2(n)q_{i(n,d)} \cong F(n)$$

Again, we will prove that this penalty term is equal to zero when all the constraints are satisfied. Assume that $h_1(n) = h_2(n) = F(n) = 1$.

We have $q_i(n_3, d_1) = 0$, $q_i(n_3, d_2) = 0$ which means that nurse_3 is not scheduled to work. Thus, given that nurse_3 is not assigned to any shifts, we will proceed with the equation without this variable. Therefore, we will solve the equation for only nurse =2 and days =2.

$$\begin{aligned} & \gamma \sum_n^2 \left(\sum_d^2 h_1(n)h_2(d)q_{i(n,d)} - F(n) \right)^2 = \\ & \gamma \sum_n^2 \left((h_1(n)h_2(d_1)q_{i(n,d_1)} + h_1(n)h_2(d_2)q_{i(n,d_2)}) - F(n) \right)^2 = \\ & \gamma \sum_n^2 \left([h_1(n)h_2(d_1)q_{i(n,d_1)}]^2 + [h_1(n)h_2(d_2)q_{i(n,d_2)}]^2 + [F(n)]^2 \right. \\ & \quad + 2[h_1(n)h_2(d_1)q_{i(n,d_1)}][h_1(n)h_2(d_2)q_{i(n,d_2)}] \\ & \quad \left. - 2[h_1(n)h_2(d_2)q_{i(n,d_2)}]F(n) - 2[h_1(n)h_2(d_1)q_{i(n,d_1)}]F(n) \right) = \\ & \gamma \left([h_1(n_1)h_2(d_1)q_{i(n_1,d_1)}]^2 + [h_1(n_1)h_2(d_2)q_{i(n_1,d_2)}]^2 + [F(n_1)]^2 \right. \\ & \quad + 2[h_1(n_1)h_2(d_1)q_{i(n_1,d_1)}][h_1(n_1)h_2(d_2)q_{i(n_1,d_2)}] \\ & \quad - 2[h_1(n_1)h_2(d_2)q_{i(n_1,d_2)}]F(n_1) - 2[h_1(n_1)h_2(d_1)q_{i(n_1,d_1)}]F(n_1) \\ & \quad + [h_1(n_2)h_2(d_1)q_{i(n_2,d_1)}]^2 + [h_1(n_2)h_2(d_2)q_{i(n_2,d_2)}]^2 + [F(n_2)]^2 \\ & \quad + 2[h_1(n_2)h_2(d_1)q_{i(n_2,d_1)}][h_1(n_2)h_2(d_2)q_{i(n_2,d_2)}] \\ & \quad \left. - 2[h_1(n_2)h_2(d_2)q_{i(n_2,d_2)}]F(n_2) - 2[h_1(n_2)h_2(d_1)q_{i(n_2,d_1)}]F(n_2) \right) \\ & q_{i(n_1,d_1)} = 1 \\ & \underline{\underline{q_{i(n_2,d_2)} = 1}} \end{aligned}$$

$$\gamma \left([h_1(n_1) h_2(d_1) \cdot 1]^2 + F(n_1)^2 - 2[h_1(n_1) h_2(d_1) \cdot 1] F(n_1) \right. \\ \left. + [h_1(n_2) h_2(d_2) \cdot 1]^2 + F(n_2)^2 - 2[h_1(n_2) h_2(d_2) \cdot 1] F(n_2) \right) \\ \xrightarrow{h_1(n) = h_2(n) = F(n) = 1} \\ \gamma (1^2 + 1^2 - 2 + 1^2 + 1^2 - 2) = 0$$

Again, we proved that when the given constraints are satisfied, the quadratic penalty for nurses becomes equal to zero. This means that the objective function, which is to minimize the penalty, is successfully achieved. In other words, our demonstration has proved that satisfying the constraints plays a crucial role in attaining the optimal solution.

D-Wave Quantum Annealer

Today, D-Wave is the leading commercial provider of quantum annealers. We sent our problem to D-Wave Leap, a solver that handles all interactions with the QPU. We implement the NSP problem in the D-wave with the QUBO model using the Hybrid Quantum Solver, which utilizes heuristic methods in combination with the annealer. D-Wave has built quantum annealers that solve optimization problems by transferring the original optimization problem to a hardware connected graph that allows nearest neighbor interactions of qubits. Thus, the D-wave quantum processing unit (QPU) can be described as a lattice of interconnected qubits. To be precise, the D-Wave QPU is not fully connected. While some qubits are connected with others with couplers, the interconnection of the qubits happens with some topologies. The current topologies available that will be used in the execution of the optimization problem are Pegasus (Advantage QPUs) and Zephyr (next-generation QPUs).

To understand the topologies, it is useful to mention the categories of the couplers. There are typically three types of couplers that we encounter: internal, external, and odd. Internal couplers help connect pairs of orthogonally oriented qubits, while external couplers join pairs of parallel qubits in either the same row or column. Similarly, aligned

pairs of qubits are connected by odd couplers.

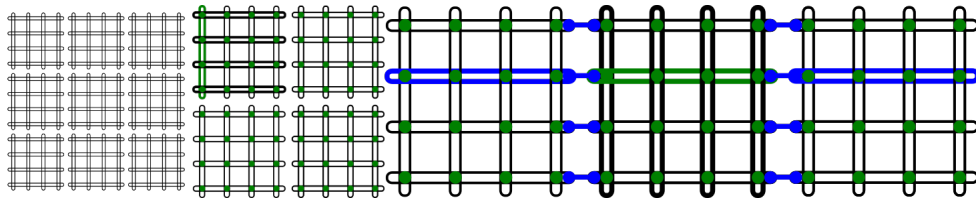


Figure 3.5: Left: the qubit orientation for 72 qubits (36 horizontal and 36 vertical). Center: Vertical green qubits couples internally with 4 horizontal black qubits. Right: Horizontal green qubits couples externally with 2 horizontal blue qubits in adjacent unit cells. Source

D-Wave's documents tell us that Qubits in Pegasus topology are arranged either vertically or horizontally, and qubits that are similarly oriented are also shifted.

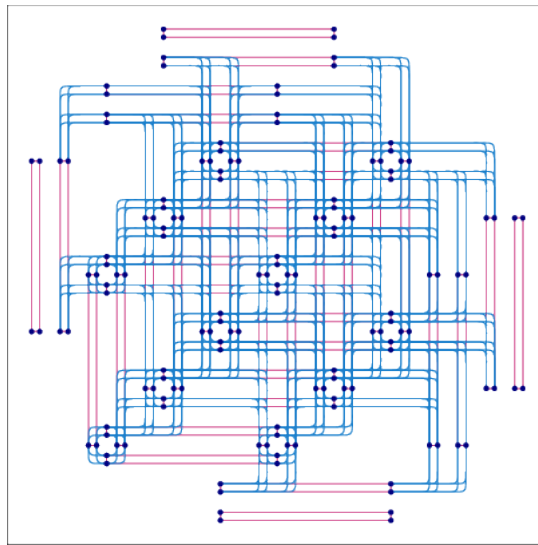


Figure 3.6: Coupled qubits roadway-style in a 144 nodes-sized Pegasus processor. Qubits: dots, couplers: lines. Curved blue lines represent "internal" couplers, while long red lines represent "external" couplers and short red lines represent "odd" couplers.

The Pegasus topology involves connecting every qubit to 12 orthogonal qubits via internal couplers while also coupling it to 15 different qubits.

In the Zephyr topology, qubits are arranged either vertically or horizontally, just like in the Pegasus topology. Similarly, they are connected using three types of couplers. However, this topology surpasses Pegasus in terms of connections and coupling, achieving 16 connections between each qubit and orthogonal qubits and 20 couplings between different qubits.

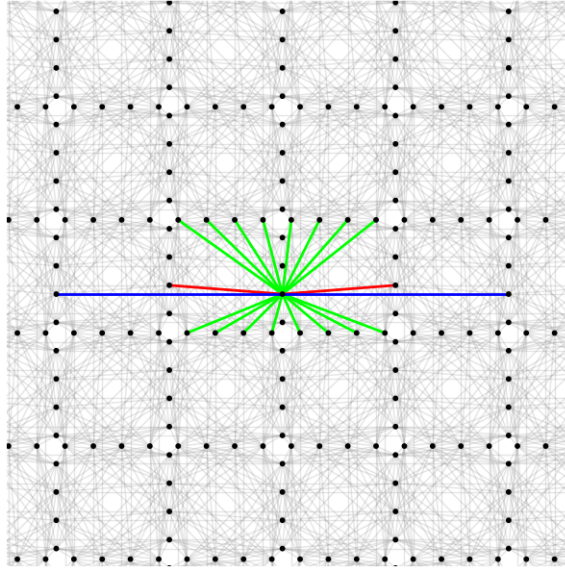


Figure 3.7: Zephyr topology. Representative qubit (black dot) connected to orthogonal qubits by 16 internal couplers (green lines). External couplers (blue lines), odd couplers (red lines)

The main difference between the two topologies is the number of connections each qubit has to other qubits. The more connections a qubit has, the more complex problems it can solve, and the less need there is for minor embedding.

It should be noted that one of the challenges in the field is the dissimilarity in connectivity between qubits in a quantum annealer and the problem at hand, which could limit the range of problems that can be solved. As we have previously discussed, the QUBO model can be represented as a graph in which interactions between spins or nodes are connected by edges. These edges encode the interactions between the qubits. However, the current topologies are restricted to nearest-neighbor connections or a few qubits rather than being fully connected. To address these limitations, we can modify the structure of the QUBO problem so that it fits within the constrained topology of the quantum annealer. This process, known as minor embedding, involves mapping the problem's connectivity to the device's connectivity.

In Figure 3.8, we can observe the mapping of a 3-variable problem on the D-Wave 2000Q processor. An extra qubit is added to address the connectivity problem.

In quantum computing, each binary variable is represented by a chain of physical qubits in the QPU. The goal is to make sure that all the physical qubits in a specific

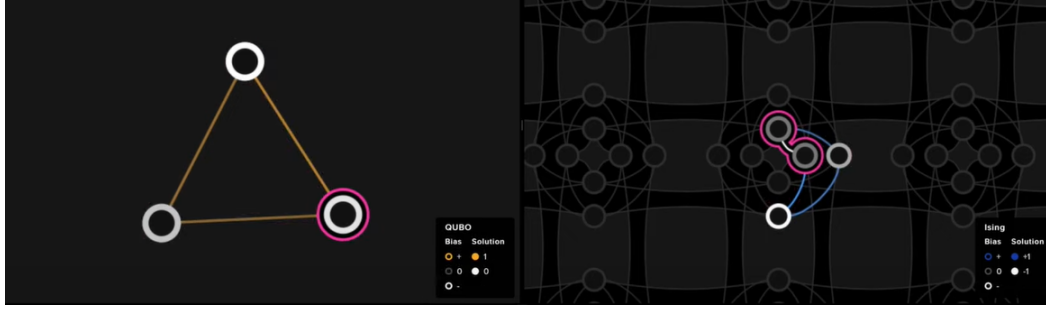


Figure 3.8: Left: The graph of QUBO problem with 3 variables, Right: The embedded problem in QPU that uses 4 qubits

chain have the same value at the conclusion of the annealing cycle. If this is achieved, it becomes simple to map the qubit values back to the variable value in the original problem. When the physical qubits in a chain have different values, post-processing software such as chain break fixing is necessary to determine which is the correct assignment to the variable. If the chains break, the solution may be degraded and less likely to be optimal. The only parameter that ensures the qubit chains have the same value is the chain strength, which is the weight assigned to the edges. If the chain strength is not large enough, the physical qubits in the chain may have different values, causing the chain to break.

Results

For our results, we set the number of collected samples per problem equal to 1000 instances, and we set up the parameters in our objective function; for hard nurse constraints, the penalty is fixed at $\alpha = \frac{7}{2}$, for hard shift constraint we set $\lambda = 1.3$, and for simplicity $E(n) = 1$ and $W(d) = 1$. Finally for soft nurse constraint we set $\gamma = 0.3$ and $h_1(n) = 1$, $h_2(n) = 1 \Rightarrow G(n, d) = 1$ and for the duty days we put $F(n) = \lfloor \frac{D}{N} \rfloor$ (Ikeda, Nakamura, and Humble).

We first create a schedule for $N = 3$ and $D = 4$ that satisfies all the constraints that were mentioned and, thus, is an optimal solution with minimized energy. In the figure 3.9, we show the graphical representation of the schedule of the Nurse Scheduling Problem. Then, we generate schedules for $N = 3$ and $D = 4$ again, but this time we alter the penalty parameter values. The results that we obtained for smaller values

sometimes meet the constraints, but in most cases, there are violations in the schedules, and the energy is not minimized.

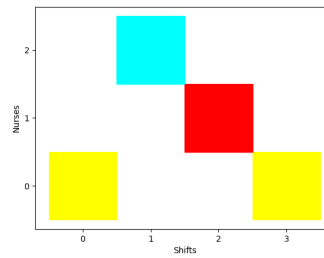


Figure 3.9: A schedule that satisfies all the constraints of this NSP instance.

In the figure 3.10, we can see that the schedules fail to meet all the required constraints. The result on the left side (lower values for parameter α) does not comply with the hard nurse constraint, which ensures that each nurse should have a period of rest between shifts. On the other hand, the result on the right side (lower values for parameter λ) violates the hard shift constraint, which requires at least one nurse to be assigned to work each day. It is worth noting that higher values of the parameters also fail, as they penalize the functions excessively, leading to the loss of good enough solutions.

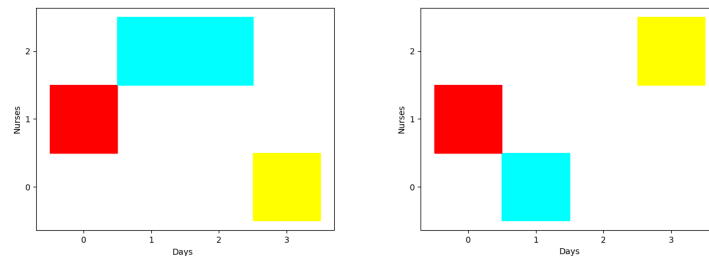


Figure 3.10: Schedules that do not satisfy all the constraints.

For this experiment, we obtain solutions for schedules with $N = 2$, $N = 3$ and $N = 4$ over a series of days, i.e. $D = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$. Since having only one nurse cover all the shifts would not be practical, we started with two nurses. We chose to increase the day parameter of the schedule as it would be more practical to arrange a small team of nurses over approximately two weeks to observe the results.

However, one may ask why we did not calculate schedules with more nurses and days. As mentioned, the nurse scheduling problem belongs to the complexity class NP.

To illustrate, let us consider a toy problem of $N = 2$ and $D = 2$. Each nurse can either work (we denote it with 1) or not work (we denote it with 0), so for every nurse, we have $2^2 = 4$ possible states (Table 3.1). Even for a minor problem such as this, we have a total of $2^4 = 16$ possible assignments demonstrating the complexity of the issue at hand (Table 3.1).

$N_{1,2}$ work both days	11
$N_{1,2}$ work at D_1 , but not on D_2	10
$N_{1,2}$ work at D_2 , but not on D_1	01
$N_{1,2}$ don't work either day	00

Table 3.1: Possible states for 2 Nurses

$N_1: 00$	$N_2: 00$
$N_1: 00$	$N_2: 01$
$N_1: 00$	$N_2: 10$
$N_1: 00$	$N_2: 11$
$N_1: 01$	$N_2: 00$
$N_1: 01$	$N_2: 01$
$N_1: 01$	$N_2: 10$
$N_1: 01$	$N_2: 11$
$N_1: 10$	$N_2: 00$
$N_1: 10$	$N_2: 01$
$N_1: 10$	$N_2: 10$
$N_1: 10$	$N_2: 11$
$N_1: 11$	$N_2: 00$
$N_1: 11$	$N_2: 01$
$N_1: 11$	$N_2: 10$
$N_1: 11$	$N_2: 11$

Table 3.2: Possible Assignments for a problem with 2 Nurses and 2 Days

It is important to note that not all possible states may be acceptable for solutions to our problem. Therefore, our next step is to calculate the states using our objective function.

$$H(q) = \sum_{n,n'}^N \sum_{d,d'}^D J_{i(n,d),j(n',d')} q_{i(n,d),j(n',d')} + \lambda \sum_d^D \left(\sum_n^N E(n) q_{i(n,d)} - W(d) \right)^2$$

$$+ \gamma \sum_n^N \left(\sum_d^D h_1(n) h_2(n) q_{i(n,d)} - F(n) \right)^2$$

$$\begin{aligned}
&= J_{i(1,1),j(1,2)}q_{i(1,1),j(1,2)} + J_{i(1,1),j(2,1)}q_{i(1,1),j(2,1)} + J_{i(1,1),j(2,2)}q_{i(1,1),j(2,2)} + \\
&\quad J_{i(1,2),j(2,1)}q_{i(1,2),j(2,1)} + J_{i(1,2),j(2,2)}q_{i(1,2),j(2,2)} + J_{i(2,1),j(2,2)}q_{i(2,1),j(2,2)} \\
&\quad + \lambda \sum_d^2 (E(1)q_{i(1,d)} + E(2)q_{i(2,d)} - W(d))^2 \\
&\quad + \gamma \sum_n^2 (h_1(n)h_2(n)q_{i(n,1)} + h_1(n)h_2(n)q_{i(n,2)} - F(n))^2 \\
&= J_{i(1,1),j(1,2)}q_{i(1,1),j(1,2)} + J_{i(1,1),j(2,1)}q_{i(1,1),j(2,1)} + J_{i(1,1),j(2,2)}q_{i(1,1),j(2,2)} + \\
&\quad J_{i(1,2),j(2,1)}q_{i(1,2),j(2,1)} + J_{i(1,2),j(2,2)}q_{i(1,2),j(2,2)} + J_{i(2,1),j(2,2)}q_{i(2,1),j(2,2)} \\
&+ \lambda (E(1)q_{i(1,1)} + E(2)q_{i(2,1)} - W(1))^2 + \lambda (E(1)q_{i(1,2)} + E(2)q_{i(2,2)} - W(2))^2 \\
&\quad + \gamma (h_1(1)h_2(1)q_{i(1,1)} + h_1(1)h_2(1)q_{i(1,2)} - F(1))^2 + \\
&\quad \gamma (h_1(2)h_2(2)q_{i(2,1)} + h_1(2)h_2(2)q_{i(2,2)} - F(2))^2
\end{aligned}$$

But according to

$$J_{i(n,d),j(n',d')} = \begin{cases} a : & \text{if } n' = n \text{ and } d' = d + 1 \\ 0 : & \text{otherwise} \end{cases}$$

$$\text{we have that } \begin{cases} \left\{ \begin{array}{l} J_{i(1,1),j(2,1)} = 0 \\ J_{i(1,2),j(2,2)} = 0 \\ J_{i(1,1),j(2,2)} = 0 \\ J_{i(1,2),j(2,1)} = 0 \end{array} \right. & \text{and} & \left\{ \begin{array}{l} J_{i(1,1),j(1,2)} = \alpha \\ J_{i(2,1),j(2,2)} = \alpha \end{array} \right.
\end{cases}$$

We will now proceed with substituting the given values for the parameters in the equation provided at the beginning of the chapter.

$$\begin{aligned}
H(q) &= \frac{7}{2}(q_{i(1,1),j(1,2)} + q_{i(2,1),j(2,2)}) \\
&+ 1,3 (1 \cdot q_{i(1,1)} + 1 \cdot q_{i(1,2)} - 1)^2 + 1,3 (1 \cdot q_{i(1,2)} + 1 \cdot q_{i(2,2)} - 1)^2
\end{aligned}$$

$$+0,3 (1 \cdot 1 \cdot q_{i(1,1)} + 1 \cdot 1 \cdot q_{i(1,2)} - 1)^2 + 0,3 (1 \cdot 1 \cdot q_{i(2,1)} + 1 \cdot 1 \cdot q_{i(2,2)} - 1)^2$$

So, the following is the objective function that we want to minimize.

$$H(q) = \frac{7}{2}(q_{i(1,1),j(1,2)} + q_{i(2,1),j(2,2)}) + 1,3 \left[(q_{i(1,1)} + q_{i(1,2)} - 1)^2 + (q_{i(1,2)} + q_{i(2,2)} - 1)^2 \right] \\ + 0,3 \left[(q_{i(1,1)} + q_{i(1,2)} - 1)^2 + (q_{i(2,1)} + q_{i(2,2)} - 1)^2 \right]$$

The solution to the problem at hand lies in identifying the optimal values of $q_{i(1,1)}, q_{i(1,2)}, q_{i(2,1)}, q_{i(2,2)}$ that minimize the function.

3.2.3 Quantum Annealing at Play

We implemented the problem with Forward Annealing first and then with Reverse Annealing by using the computed schedules from Forward. We also present the results that we obtained by using two different topologies of D-Wave QPU: the Pegasus (Advantage QPUs) and the Zephyr (next-generation QPUs), and we compared the results.

To begin, we first frame the nurse scheduling problem as a QUBO, enabling us to mathematically represent the binary optimization problem while taking into account the constraints. The QUBO is then translated onto the architecture -physical qubits and interactions- of the D-Wave quantum annealer (which utilizes a topology such as Pegasus or Zephyr) through an embedding process. Following this, we convert the QUBO matrix into a Binary Quadratic Model (BQM) for the DWaveSampler to utilize. We then execute the Forward or Reverse annealing process to generate samples of potential solutions. In this manner, the D-Wave machine employs quantum annealing to identify the ground state of the given problem Hamiltonian.

Forward Annealing

Forward annealing involves starting with a quantum system in a known and easy-to-prepare initial quantum state, where all qubits are in a superposition of 0 and 1. The system then slowly evolves from this initial superposition toward a final Hamiltonian representing the problem to be solved. The ultimate goal is for the system to be in the

ground state of the problem Hamiltonian by the end of the annealing process, which corresponds to the optimal solution.

We will present the results for the forward annealing.

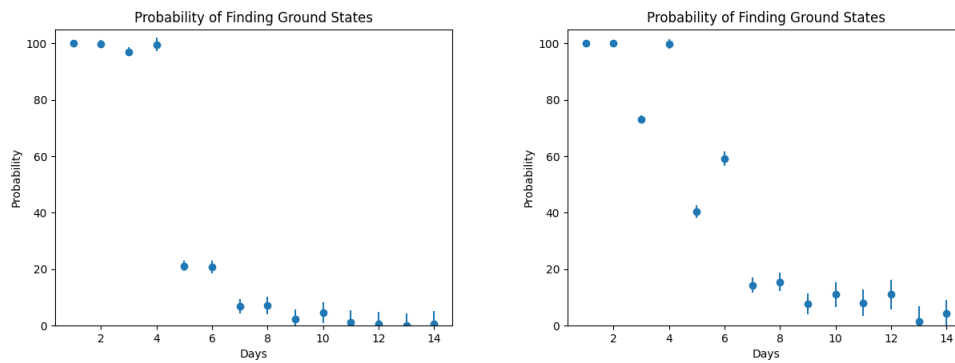


Figure 3.11: For the case of $N=2$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).

What we notice from is that for simpler schedules with smaller instances, the probability of obtaining satisfying solutions that satisfy the constraints are high, while for more complex schedules with larger instances, the probability decreases. Especially for $N = 2$, $D = 1, 2, 3, 4$ with Pegasus topology, we obtain a ground state with almost 100% probability.

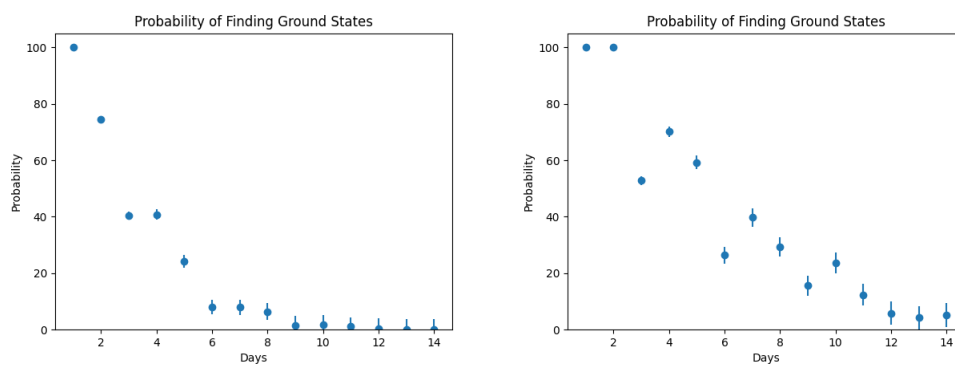


Figure 3.12: For the case of $N=3$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).

About the results that we got by using the D-Wave's Zephyr topology, we notice that for most cases, ground states were obtained. This indicates that the QUBO

problem was efficiently formulated, with the penalty terms appropriately balanced to not overwhelm the solution space. The soft constraints effectively led to more favorable solutions, ensuring that the constraints were satisfied while the annealer found optimal solutions. The quadratic and linear terms were well-suited for the quantum annealing process. Additionally, this suggests that the physical constraints of the Zephyr topology did not introduce significant issues that could hinder the identification of the optimal solution. Specifically speaking for $N = 2$, $D = 9, 10, 11, 12, 14$ and for $N = 3$, $D = 9, 10, 11, 12, 13, 14$ the results we acquire are significantly better than the almost zero probabilities of the Pegasus results. Even for smaller schedules such as $N = 2$, $D = 5, 6, 7, 8$ and for $N = 3$, $D = 5, 6, 7, 8$ the probability of finding the ground state is noticeably better.

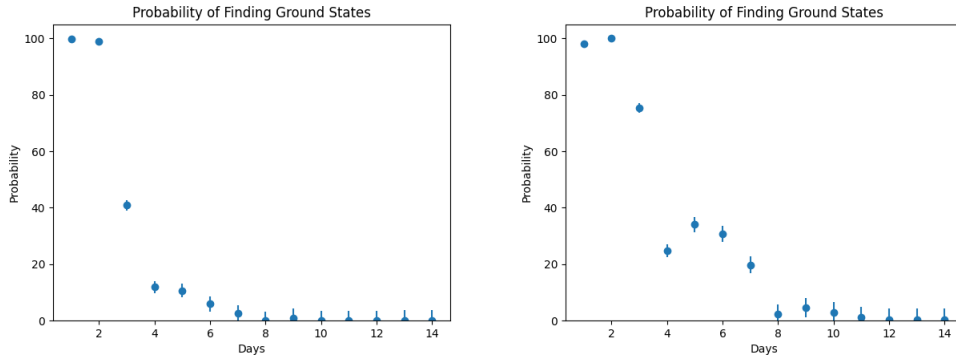


Figure 3.13: For the case of $N=4$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).

Specifically speaking the solutions for $N = 2$, $D = 13, 14$, for $N = 3$, $D = 12, 13, 14$ and for $N = 4$, $D = 8, 10, 11, 12, 13, 14$ that were obtained with D-Wave's Pegasus topology was not satisfying while a few good solutions were obtained for $N = 2$, $D = 11, 12$, for $N = 3$, $D = 9, 11$ and for $N = 4$, $D = 9$.

Reverse Annealing

We now implement the reverse annealing process for the nurse scheduling problem.

We know that with forward annealing, the quantum process starts from a random, easy-to-prepare state (usually superposition) and then explores the solution space to

find an optimal solution. On the other hand, with reverse annealing, the process starts from an already known (good) solution and then explores the neighboring solution space by making small changes. Firstly, the system is annealed backward into a quantum superposition to explore different states, allowing it to escape local minima. Then, the system starts evolving forward, moving towards the problem Hamiltonian, refining the solution, and ideally settling into a better solution than the initial guess.

We studied the case where the reverse annealing process starts with the initial state from solutions computed by forward annealing. To elaborate further, the solution that is used as the starting point for the reverse annealing is the best solution found in a prior run of forward annealing. This process, as we see from the following results, can be efficient and can improve the frequency with which a computed solution satisfies the problem constraints.

Here, we mention briefly the basic steps that we followed. First, we define the parameters and constraints of the problem. Next, we construct the Hamiltonian, and we embed the problem onto the quantum annealer while specifying the topology that we want to use (Pegasus or Zephyr). Then, the annealing process starts by loading the initial state from a previous forward annealing run. The following procedure is evolving the system from the initial state to an intermediate state, pausing for a hold period, and then evolving to the final state. Finally, we unembed the solutions.

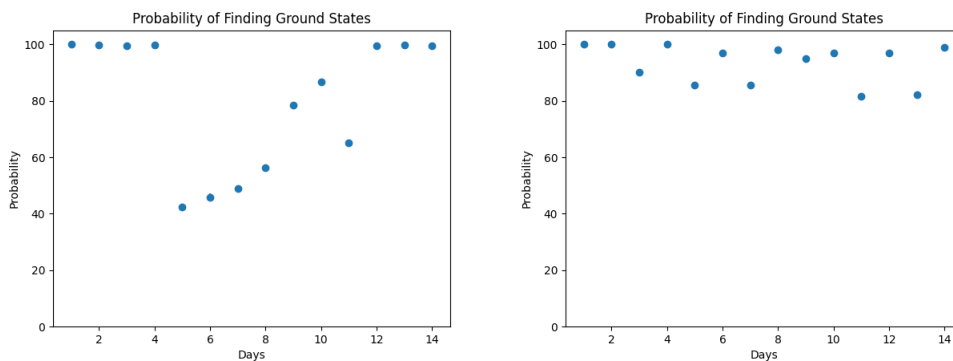


Figure 3.14: For the case of $N=2$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state by Reverse Annealing using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).

We observe that for $N = 2$ and $D = 1, 2, 3, 4, 12, 13, 14$ using Pegasus topology and for

$N = 2$ and $D = 1, 2, 4, 8, 14$ using Zephyr topology the results appear to have nearly 100% probability to obtain a good solution.

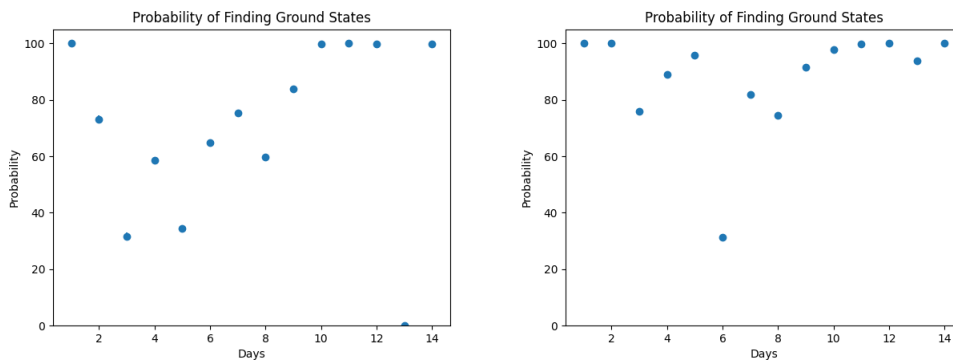


Figure 3.15: For the case of $N=3$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state by Reverse Annealing using D-Wave's QPU with two different topologies: Pegasus (left) and Zephyr (right).

From the plots, we see that for different days, the probability of finding a ground state varies. It is worth mentioning that for $N = 3$ and $D = 13$ the probability decreases to zero. This could be because of a problem with the annealing process or with the initial state that is used for the process, or it could just be that this specific problem is too difficult to solve.

What we notice from the results shown using Pegasus topology for $N = 2$ and for $N = 3$ and with Zephyr topology for $N = 3$ is that for $D = 1$ the probabilities are close to 100%, then we notice a drop off the probabilities, and after that, a recovery follows and the probabilities go up to almost 100%. This could indicate that the particular parameters and characteristics of each problem affect the results of finding the ground states.

We notice that for $N = 4$ and $D = 10, 11, 12, 13, 14$ the probability of finding the ground state using the Pegasus topology is equal to zero. What we notice is that, in general, the reverse annealing process improves the computed solutions for most of the cases. Specifically, the results obtained by using the Zephyr topology appeared to be better.

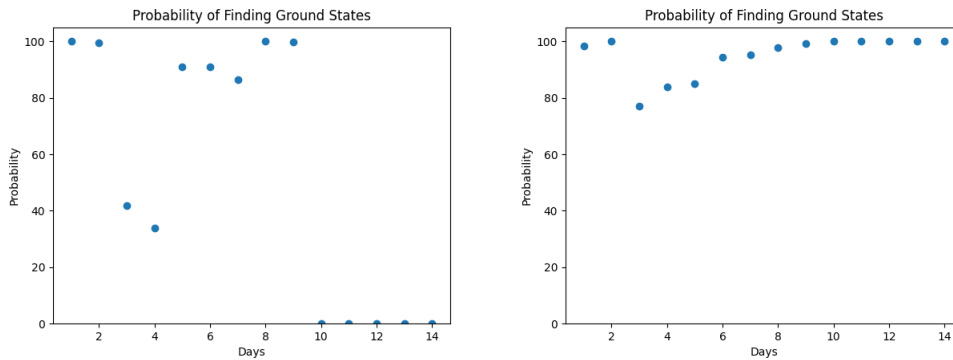


Figure 3.16: For the case of $N=4$ over a schedule of D days $D=[1,14]$: the probability of finding a ground state by Reverse Annealing using D-Wave's QPU with two different topologies: Pegasus(left) and Zephyr(right).

3.2.4 Solution Checking

We want to evaluate the computed solutions that we get and compare them with each other and with the completely satisfying solutions. To do that, we use the Hamming Distance. In most cases, the Hamming Distance is a measure of the difference between two strings of equal length. In the context of binary vectors, Hamming Distance is the number of bit positions at which the code differs. What we did for the calculations was to represent the schedules as ordered binary vectors of size ND , where N is the number of nurses and D is the number of days. Each element in the vector represents whether a particular nurse is working on a particular day, with one meaning that is working and zero meaning that is not. We then took these binary vectors, and we measured the number of positions at which they agree by using the exclusive OR operation that gives the number of positions at which two bits are different (returns 1 if two bits are different and 0 if they are the same). In that way, we calculated the Hamming distance between consecutive states (solutions), and we practically measured how much each solution differed from the next one. With that being said, it is apparent that the maximum value of the Hamming distance is ND , and it would occur only when we compare two schedules that are entirely different, while on the other side, the minimum value is zero with that meaning that the schedules are the same. In the figures below, we plot the means that we obtained from D-Wave and the standard deviation (error bars) of the Hamming distances between consecutive states.

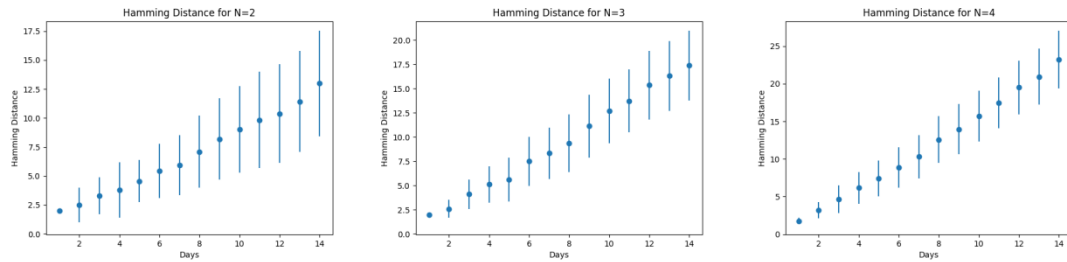


Figure 3.17: The Hamming distance between consecutive Forward Annealing solutions concerning a number of days for $N=2$ nurses (left), $N=3$ nurses (middle), and $N=4$ nurses (right), with Pegasus.

By examining the plots of Hamming distances for forward annealing, we notice that as the scheduling problem becomes more complex, i.e., the number of days increases, and the Hamming distance increases, too. This means that the solutions generated by the quantum annealing process become more different from each other as the problem becomes more complex. The Hamming distance seems to increase linearly as the number of days increases. This could be due to the fact that as the number of days increases, the number of possible schedules also increases, resulting in a larger space of potential solutions. This more significant variation in the possible solutions may be causing the observed increase in the Hamming distance (as a measure of the difference between the possible schedules). From the plot, we see the rate of increase, i.e., how much the Hamming distance increases per day, appears to be different for different values of N , which suggests that the variation of the solutions also depends on the number of nurses.

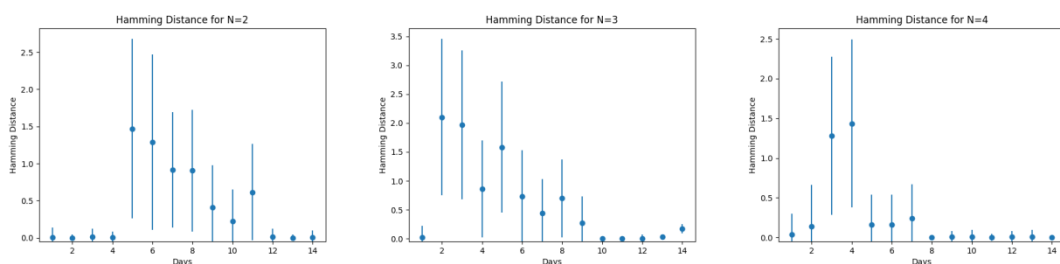


Figure 3.18: The Hamming distance between consecutive Reversed Annealing solutions with respect to some scheduled days for $N=2$ nurses (left), $N=3$ nurses (middle), and $N=4$ nurses (right), with Pegasus.

By looking at figure 3.18, we observe that the Hamming distance is relatively small in all three cases of reverse annealing. For certain values of days, it is even zero or close

to zero. This can indicate that the solutions are quite similar to each other, and the solutions are successfully refined.

Comparing the plots of the Hamming distances between consecutive reverse annealing solutions and those between forward annealing, we notice that reverse annealing results are significantly lower compared to the results from forward annealing. This could be due to the differences in how these two processes explore the solution space. As we have already pointed out, the process of reverse annealing uses the computed schedules from forward annealing. It starts from a known (good) solution and makes only minor changes. Due to this approach, it is highly likely that the results obtained from reverse annealing will be similar to each other. This makes sense if we think that, in comparison, the forward annealing starts from a random solution and then searches for the optimal solution by expanding the solution space, so it is only logical solutions that are obtained from the process to be more diverse.

We also calculated the fraction of the solutions that are in the ground state for each combination of nurses and days. Here are the results we obtained.

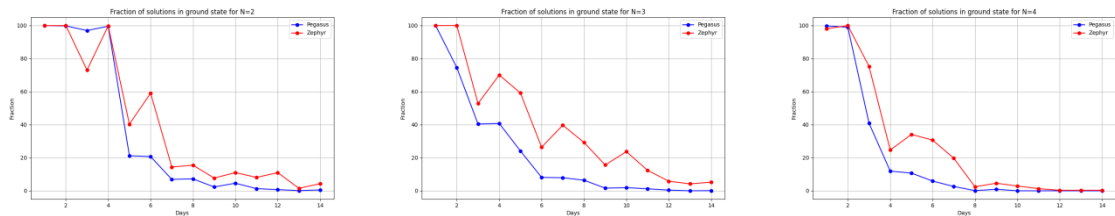


Figure 3.19: The fraction of solutions that are in the ground state for the NSP with Forward Annealing by using the Pegasus topology (blue) and the Zephyr topology (red).

What we notice from the figure 3.19 is that the percentage of finding the optimal solution for the problem decreases as the number of days increases, which is what we expected since the problem gets more complex. However, we also observe that the fraction also depends on the topology. The Zephyr topology appears to perform better than the Pegasus, so we could suggest that the problem with these values and size is slightly easier to solve for the Zephyr topology. Upon a close examination of the fractions, it becomes apparent that Zephyr topology never reaches exactly 0%, when at the same time Pegasus topology drops to 0% for $N = 3$ and $D = 13$ and for $N = 4$ and $D = 10, 11, 12, 13, 14$.

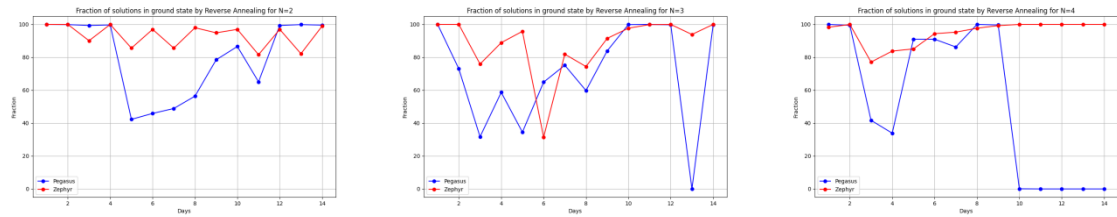


Figure 3.20: The fraction of solutions that are in the ground state for the NSP with Reverse Annealing by using the Pegasus topology (blue) and the Zephyr topology (red).

As for the results obtained with reverse annealing, we see that the number of good solutions to the number of total solutions appears to be high. We compared the results with the ones from forward annealing, and they seem to be significantly improved, which aligns with the previous results. Furthermore, after analyzing the diagrams for Zephyr topology with $N = 3$ and $N = 4$, we noticed the fact that the results are good for small instances of days, but then they drop. However, after a point, as the number of days increases, the results seem to be good again. This behavior is not as straightforward as it is for forward annealing, where the more complex the problem, the lower the percentage. It could indicate that the results are impacted directly by the characteristics of each problem.

Chapter 4

Conclusion and Future Work

In our thesis, we presented various concepts for solving binary optimization challenges. Here, we summarize our key findings and future prospects.

The first chapter delves into the fundamental principles of quantum computing, introducing qubits as the building blocks of quantum information processing. It also discusses qubits' unique properties, such as superposition and entanglement, and analyzes the potential computational power of quantum systems using the Deutsch-Jozsa algorithm.

The second chapter focuses on the transition from classical to quantum optimization, examining combinatorial optimization and its applications through Quantum Adiabaticity, Quantum Annealing, and Variational Quantum Algorithms. It further discusses techniques for solving optimization problems, including the Quantum Approximate Optimization Algorithm, and attempts to solve the MaxCut problem with the algorithm. We also discussed the Quadratic Unconstrained Binary Optimization and its connection to the Ising model, emphasizing the importance of addressing these models for tackling combinatorial optimization problems in quantum computing.

The third chapter highlights the application of quantum computing to scheduling problems. Specifically, we formulated the Nurse Scheduling Problem (NSP) as a Hamiltonian into QUBO form and solved it for some instances using quantum annealing with the cloud provided by D-Wave. The goal is to estimate the probability of satisfying all constraints and explore the results and potential of quantum computing in solving scheduling problems.

Our findings show that as the schedule size increases, i.e., the number of days and nurses, the probability of successfully matching all constraints decreases for a fixed sample size and annealing duration. This is expected because larger problems have a bigger search space, making it more difficult to find the optimal solution. The results also indicate that although reverse annealing methods improve the probability of success, the improvement is not uniform.

While quantum computation is still in its early stages, the approach presented may be extended to more general scheduling problems. Additionally, researchers anticipate advancements in quantum hardware and software technologies, which is especially important in the Noisy Intermediate Scale Quantum (NISQ) era, in which we are now, where quantum computers have considerable error rates and limited sizes due to the number of logical qubits in the system. The potential impact of quantum tunneling and noise on our results is also recognized. Efforts are being made to develop robust error correction techniques and fault-tolerant architectures to overcome current challenges and ensure practical usage of quantum computers. Moreover, researchers aim to increase the number of qubits in a scalable manner. These improvements could provide the opportunity to handle realistic sizes of schedules for our problem.

There is a global curiosity about quantum computing, with researchers and companies racing to build practical quantum computers. As the field advances, quantum computers are becoming more accessible through cloud services, and their popularity continues to grow rapidly. We can confidently state that we are now in the quantum era, efforts are ongoing to overcome the remaining hurdles and unlock the full potential of quantum computing.

Appendix A

Job Shop Scheduling Problem (JSSP)

A.1 Job Shop Scheduling Problem (JSSP)

The Job Shop Scheduling problem (JSP) is a challenging combinatorial optimization problem with many real-world applications. It involves scheduling a series of operations, or routing, for each job to be processed on different machines. Due to its NP-hard complexity, this problem has been extensively researched for many years.

A.1.1 Problem Definition

The job-shop scheduling problem involves a set of operations $\mathcal{A} = \{o_1, \dots, o_{|\mathcal{A}|}\}$ that need to be scheduled on M machines. Each operation o_i has a positive integer processing time (duration) d_i . The operations are divided into N jobs A_1, A_2, \dots, A_N where $A_j \subseteq \mathcal{A}$ and $|A_j| = M$. Each job needs to be processed on each machine exactly once, so a job consists of M operations, and the number of jobs is $N = \frac{|\mathcal{A}|}{M}$.

Operations within the same job must be processed in a specific order. Particularly, for every A_j we define $\sigma^j = \langle \sigma_1^j, \sigma_2^j, \dots, \sigma_M^j \rangle$ with $\sigma_h^j \in \{1, \dots, M\}$ representing the machine that must process the h -th operation of the job. For simplicity, we denote the operation of job j on machine m as $o_{j,m}$, with its duration (execution time) $d_{j,m}$, and we use d_{\max} to represent the maximum duration of operations in \mathcal{A} .

It is crucial to ensure that no two operations overlap on the same machine since machines cannot perform more than one operation at any given time (unary capacity machines). We also assume non-preemption, i.e., an operation that has started processing cannot be interrupted, so each operation needs to be completed before the following one can start. Furthermore, the problem is also subject to the following assumptions: all machines are available at time zero, and all jobs are available at time zero. The classical objective of JSP is to schedule all operations in a valid sequence while minimizing the makespan, meaning the latest completion time of the operations (i.e., the completion time of the last running job). We use $\text{JSP}(|\mathcal{A}|, M, d_{\max})$ to represent a JSP specification. Note that execution times are often assumed to be larger than zero which is simply a generalization. By allowing execution times (durations) of zero, we add an extra constraint without changing the problem description since it does not contribute to the length of a job or the makespan directly. The extra constraint which is introduced with $d_i = 0$, is that the machine should be either idle when the operation is scheduled or about to start another operation at this very point in time.

In some scheduling problems, the due date is also considered, and the objective can also include the due date constraints. Variable s_j represents the integer start time of the operation of job j , while dd_j is the due date variable of the job. The ready time and the completion time of job j are r_j and c_j respectively. The available time to finish the job on time is calculated by $a_j = dd_j - r_j$ and the flow time of the job is $F_j = c_j - r_j$. In order for the schedule to meet job due dates, it must not be late; the lateness of a job is $L_j = c_j - dd_j$ where $L_j < 0$ means that the job is early and $L_j > 0$ means that the job is finished later than the due date and so there exist tardiness.

So, the Job Shop Scheduling Problem serves as a comprehensive framework for optimizing the allocation of resources needed to carry out a series of operations while taking into account constraints related to location and time.

Many solution approaches for the JSP have been suggested in the literature. These include the well-known branch-and-bound approach of Carlier & Pinson (1989) and the shifting bottleneck heuristic of Adams et al. (1988). Both of these methods depend on effectively solving single-machine scheduling problems. In general, simple and fast

heuristics rely on dispatching rules that are tailored to the specific problem. The dispatching heuristics provide quick and easy-to-understand schedules that require relatively small computation time. The disadvantage of dispatching rules is that these cannot hope for optimal solutions because the decision is made only according to the local information.

The use of Dispatching Rules to select a job from available jobs for processing is a common practice. By changing the dispatching rule, the schedule also changes because each rule prioritizes different jobs for processing. Several dispatching rules exist to specify the order in which jobs should be completed at each machine. The first category is the one that is based on the processing time: Shortest Processing Time (SPT), Longest Processing Time (LPT), Shortest Remaining Processing Time (SRPT), Longest Remaining Processing Time (LRPT), Shortest Total Processing Time (STPT), Longest Total Processing Time (LTPT). Another category is the due date based rules: Earliest Due Date (EDD), Modified Due Date (MDD), and Slack, which is calculated using the formula: $Slack = Due_date - current_time - remaining_processing_time$. Additional rules include First In First Out (FIFO), Last In First Out (LIFO), random selection, and a combination of rules.

Job Shop Scheduling Problem - Example

To better understand the formulation and the goal of the JSP problem, let us solve an example.

Suppose we have three jobs and three machines, with each job having a predefined route for visiting the machines as indicated in table A.1. The schedule is non-delay, so the machine should not stay idle while jobs are waiting. For this problem, we will use the SPT dispatching rule, which involves selecting the job with the shortest processing time, as mentioned earlier.

J1	M1(7)	M3(8)	M2(10)
J2	M2(6)	M1(4)	M3(12)
J3	M1(8)	M2(8)	M3(7)

Table A.1: Route of machines

Note that in table A.1, we see the processing times denoted within brackets, which represent the processing times for each job on the respective machine.

The following chart (figure A.1) is called the Gantt chart. It was designed by Henry Gantt, who first proposed a pictorial representation of a schedule around 1910-1915. It is essentially a bar chart that illustrates a project schedule. Once drawn, evaluating objectives is simple. However, it's important to note that although the Gantt chart is a valuable evaluation tool, it does not guarantee optimality.

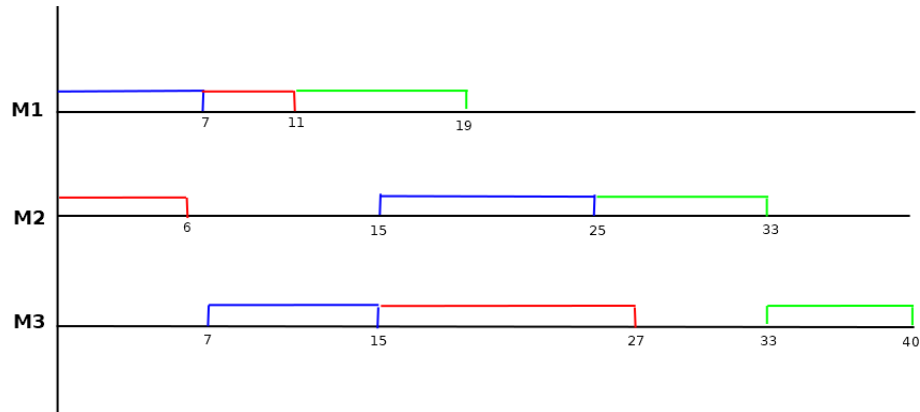


Figure A.1: Problem solution in Gantt Chart.

Notice that to choose among the three jobs that are waiting, we would pick the job with the smallest processing time. So, among jobs J1, J2, and J3, we start with J2, which has a processing time of 6. Then, among J1 and J3, both of which initially require the machine 1, we choose to start with J1, which has a processing time of 7. After seven time units, J1 finishes with M1. Now, both J3 and J2, which have already finished from M2 one time unit earlier, wait to be processed in the first machine. Once again, J2 has the shortest processing time, needing 4 time units compared to 8 for J3. Therefore, J2 follows J1 in M1. The last job to be processed in M1 is J3. It is important to note that machines M2 and M3 remain idle during certain time intervals, such as from time unit six to fifteen for M2 and from time zero to seven and twenty-seven to thirty-three for M3, because there are no available jobs for those specific machines at those times. For example, at time units twenty-seven to thirty-three at M3, every operation in J1 and J2 has been completed, and the J3 is still being processed at M2, so M3 waits until the operation of J3 finishes from M2.

The Gantt Chart analysis reveals that J1 finishes at 25 time units, concluding its last job at M2. J2 completes at 27 time units, with its final job at M3, while J3 finishes at 40 time units, also concluding its last job at M3. (Table A.2)

Job	Completion Time
J1	25
J2	27
J3	40

Table A.2: Completion time of jobs

All the jobs are completed at a time equal to 40, so the makespan for this fixed schedule is 40. The total completion time is 92 because $\sum CT = 27 + 25 + 40$.

Now, let us consider a scenario where each job has a specific due date, as indicated in the table A.3.

Job	Completion Time	Due Date
J1	25	30
J2	27	25
J3	40	35

Table A.3: Completion time of jobs

According to the new data, we can tell that the first job finishes before the due date, so it is early. However, the second and the third jobs are late with tardiness equal to 2 and 5 respectively.

The total tardiness of the problem will be $\sum T = 7$ and the number of tardy jobs are $n_j = 2$. Last but not least, the maximum tardiness is $L_{\max} = 5$.

The Disjunctive Graph Model

The disjunctive graph proposed by Roy and Sussman is a well-known model used to describe instances of job-shop scheduling problems. The disjunctive graph is a directed graph $G = (V, C \cup D)$. V denotes a set of nodes corresponding to the operation of the jobs. The weight of each node is set to be the same as the processing time of the operation it represents. This set contains two extra nodes with zero processing time: the source, which is an operation proceeding with all the other operations, and the sink, which succeeds all the other operations. The notation for the source and the sink varies,

with some being 0, S and $*$, F respectively. C denotes a set of conjunctive arcs that reflect the initial constraint of respecting the job order of all the operations by connecting every two consecutive operations of the same job. As for the set D , it denotes all the undirected disjunctive edges that connect the unordered operations that require the same machine for their execution. This is usually represented by two opposite-directed arcs.

As mentioned before, the goal of the job shop scheduling problem is to find a schedule with a minimal length by finding the optimal order of all tasks on the machine. The equivalent of finding the optimal solution to the disjunctive graph involves selecting one arc in each disjunction, ensuring that the longest path from the source to the sink is minimal. The length of this longest path determines the makespan. It is important to note that the resulting graph must be acyclic.

Job Shop Scheduling Problem - Example

Now, we will represent the previously mentioned example with a graph to better understand it. For the sake of memory efficiency, please refer to the table mentioned in the preceding example (Table A.1).

We draw the representation of this example of a job shop scheduling problem with a graph.

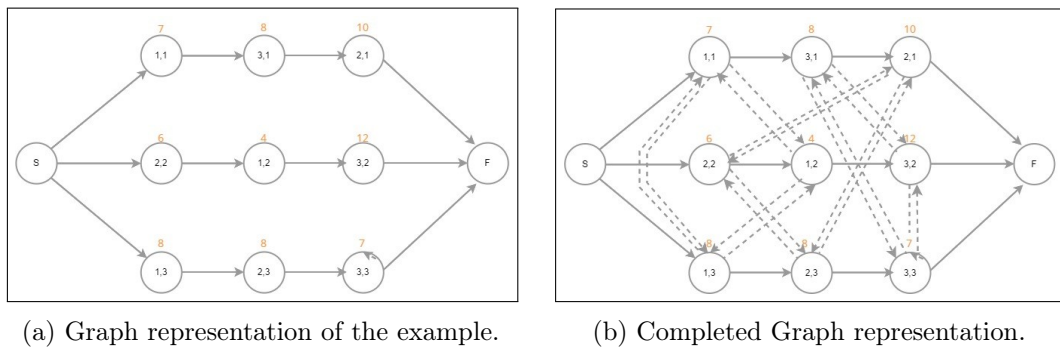


Figure A.2: Graph representation

In this representation (Figure A.2(a)), the three jobs; J_1 , J_2 , J_3 are represented as 3 paths in the graph. The nodes represent the operations of the jobs; $n(i, j)$ where i represents the machine and j represents the job. The nodes S and F correspond to the

start and the finish, respectively. The arrows indicate the order of machine visits by the operations, and the orange numbers represent the processing times.

In this representation (Figure A.2(b)), the arcs represent the paths. As we are yet to determine which machine will be active and when, we consider all possible arcs, with each arc indicating the makespan of a feasible solution.

Several classical approaches to solving the Job Shop Scheduling Problem are based on applying constraints propagation techniques. While the JSP can be expressed as a mixed integer programming (MIP) problem in different ways, such as the rank-based formulation, here we will introduce two direct QUBO formulations for JSP, which are based on two popular MIP models of JSP: the disjunctive and the time-index models.

A.1.2 Disjunctive QUBO Model

The disjunctive QUBO (DQ) model which is based on the disjunctive MIP model (Manne 1960), utilizes the variable $\hat{x}_{j,m}$ to denote the integer start time of the operation of job j on machine m , and \hat{C}_{\max} to represent the integer makespan variable. Additionally, the variable $z_{j,k,m}$ is a binary variable that

$$z_{j,k,m} = \begin{cases} 1 & \text{if the op. of job } j \text{ on machine } m \\ & \text{precedes the op. of job } k \text{ on machine } m, \\ 0 & \text{otherwise.} \end{cases}$$

$\hat{s}_{j,h}^{(1)}$, $\hat{s}_j^{(2)}$, $\hat{s}_{j,k,m}^{(3,1)}$, $\hat{s}_{j,k,m}^{(3,2)}$ are integer slack variables used in different constraints.

We represent every non-negative integer variable as its binary expansion. For instance, the binary representation of C_{\max} is a sequence of binary bits $i^{(k)} \in \{0,1\}$, where k ranges from 1 to $L(C_{\max})$. This representation is given by the equation $C_{\max} = \sum_{k=1}^{L(C_{\max})} 2^{k-1} i^{(k)}$.

To represent the value C_{\max} , we need $L(C_{\max}) = \lceil \log_2 C_{\max} \rceil + 1$ binary bits. We'll use \hat{C}_{\max} to denote the binary expansion of C_{\max} . Additionally, we'll use the notation \hat{x} to represent the binary expansion of any integer variable x .

The DQ is as follows.

$$\begin{aligned}
\min_{\hat{x}, z, \hat{s}} & \hat{C}_{\max} + p_1 \sum_{j=1}^N \sum_{h=2}^M \left(\hat{x}_{j, \sigma_{h-1}^j} + d_{j, \sigma_{h-1}^j} + \hat{s}_{j,h}^{(1)} - \hat{x}_{j, \sigma_h^j} \right)^2 \\
& + p_2 \sum_{j=1}^N \sum_{h=2}^M \left(\hat{x}_{j, \sigma_M^j} + d_{j,M} + \hat{s}_j^{(2)} - \hat{C}_{\max} \right)^2 \\
& + p_3 \sum_{m=1}^M \sum_{j=1}^{N-1} \sum_{k=j+1}^N \left(\hat{x}_{k,m} - \hat{x}_{j,m} - d_{j,m} + V - V \cdot z_{j,k,m} - \hat{s}_{j,k,m}^{(3,1)} \right)^2 \\
& + \left(\hat{x}_{j,m} - \hat{x}_{k,m} - d_{k,m} + V \cdot z_{j,k,m} - \hat{s}_{j,k,m}^{(3,2)} \right)^2
\end{aligned}$$

$$z_{j,k,m} \in \{0, 1\}, \quad \forall j, k = 1, \dots, N, j < k, \quad \forall m = 1, \dots, M,$$

$$\hat{C}_{\max}, \hat{x}_{j,m}, \hat{s}_{j,h}^{(1)}, \hat{s}_j^{(2)}, \hat{s}_{j,k,m}^{(3,1)}, \hat{s}_{j,k,m}^{(3,2)} \geq 0,$$

$$\forall k, k = 1, \dots, N, j < k, \quad \forall m = 1, \dots, M.$$

Let us analyze each term.

1. $\min_{\hat{x}, z, \hat{s}} \hat{C}_{\max}$ is the original objective function of the JSP.
2. The expression $p_1 \cdot \sum_{j=1}^N \sum_{h=2}^M \left(\hat{x}_{j, \sigma_{h-1}^j} + d_{j, \sigma_{h-1}^j} + \hat{s}_{j,h}^{(1)} - \hat{x}_{j, \sigma_h^j} \right)^2$ represents the quadratic representation of precedence constraints. This term is added to the objective function as a penalty term.

A precedence constraint ensures that the start time of the h -th operation of job j is greater than the end time of the $(h-1)$ -th operation of job j if $h > 1$. In other words,

$$\hat{x}_{j, \sigma_h^j} \geq \hat{x}_{j, \sigma_{h-1}^j} + d_{j, \sigma_{h-1}^j}.$$

We penalize the violation of this constraint by transforming it using the expression

$$p_1 \cdot \left(\hat{x}_{j, \sigma_{h-1}^j} + d_{j, \sigma_{h-1}^j} + \hat{s}_{j,h}^{(1)} - \hat{x}_{j, \sigma_h^j} \right)^2,$$

where p_1 is a sufficiently large integer.

In the context of minimizing the overall objective, we aim for the

penalty to be 0 when the constraint is satisfied and non-zero when it is violated.

$$p_1 \cdot \sum_{j=1}^N \sum_{h=2}^M \left(\hat{x}_{j,\sigma_{h-1}^j} + d_{j,\sigma_{h-1}^j} + \hat{s}_{j,h}^{(1)} - \hat{x}_{j,\sigma_h^j} \right)^2 = \begin{cases} 0 : & \text{satisfied,} \\ \text{non-zero} : & \text{not satisfied.} \end{cases}$$

Hence, we introduce an integer slack variable to the penalty, and it becomes the original term. This term ensures that the overall objective is significantly increased if the precedence constraints are violated, which QUBO solvers would reasonably seek to avoid as long as p_1 is sufficiently large.

3. Similarly, term $p_2 \cdot \sum_{j=1}^N \sum_{h=2}^M \left(\hat{x}_{j,\sigma_M^j} + d_{j,M} + \hat{s}_j^{(2)} - \hat{C}_{\max} \right)^2$ is the quadratic representation of the makespan constraints.

The total makespan is greater or equal to the starting time of every operation plus its processing time.

$$\hat{x}_{j,\sigma_M^j} + d_{j,M} \leq \hat{C}_{\max}$$

We penalize the constraint violation by transforming it to

$$p_2 \cdot \left(\hat{x}_{j,\sigma_M^j} + d_{j,M} + \hat{s}_j^{(2)} - \hat{C}_{\max} \right)^2.$$

4. The term

$$p_3 \cdot \sum_{m=1}^M \sum_{j=1}^{N-1} \sum_{k=j+1}^N \left(\hat{x}_{k,m} - \hat{x}_{j,m} - d_{j,m} + V - V \cdot z_{j,k,m} - \hat{s}_{j,k,m}^{(3,1)} \right)^2 + \left(\hat{x}_{j,m} - \hat{x}_{k,m} - d_{k,m} + V \cdot z_{j,k,m} - \hat{s}_{j,k,m}^{(3,2)} \right)^2$$

is the quadratic representation of the disjunctive constraints, with the help of a sufficiently large constant V to enforce the disjunction.

$$\hat{x}_{j,m} + d_{j,m} \leq \hat{x}_{k,m} + V(1 - z_{j,k,m})$$

This constraint is about each machine processing just one job at a time.

A.1.3 Time-indexed QUBO Model

In the time-indexed model, we use a slightly different notation than the previous one to enhance clarity and facilitate better comprehension.

For each operation, a set of binary variables is assigned to represent specific timestamps at which the operation can start (various possible discrete starting times)

$$x_{i,t} = \begin{cases} 1 & \text{operation } O_i \text{ starts at time } t \\ 0 & \text{otherwise} \end{cases}$$

Here, the variable t represents the time and is bounded by an arbitrary deadline T , which needs to be estimated (for example, using a heuristic algorithm). This deadline T is common for all jobs and signifies the maximum allowed time for the completion of the jobs. Additionally, the deadline or timespan T is constrained by the total work involved in the problem, which is the sum of the execution times of all operations.

Let us now define a set of constraints to ensure a feasible solution and completion before time T . These various constraints are considered by adding penalty terms to the QUBO problem. The following formula is used to express the constraint that an operation must start once and once only

$$\left(\sum_t x_{i,t} = 1 \text{ for each } i \right) \rightarrow \sum_i \left(\sum_t x_{i,t} - 1 \right)^2$$

There can only be one job running on each machine at any given point in time, leading to the quadratic constraint

$$\sum_{(i,t,k,t') \in R_m} x_{i,t} x_{k,t'} = 0 \text{ for each } m$$

where R_m is a union of two sets $R_m = A_m \cup B_m$ and

$$A_m = \left\{ (i, t, k, t') : (i, k) \in I_m \times I_m, i \neq k, 0 \leq t, t' \leq T, 0 < t' - t < p_i \right\}$$

$$B_m = \left\{ (i, t, k, t') : (i, k) \in I_m \times I_m, i < k, t' = t, p_i > 0, p_j > 0 \right\}$$

with p_i denoting the processing time of operation O_i and I_m the set of all operations that have to be processed on the machine m .

The first set A_m constraints operation O_j from starting at t' if there is another operation O_i still running on the machine. This happens if O_i started at time t and $t' - t < p_i$.

The second set B_m constraints two operations to start at the same time unless one of their execution times is equal to zero.

Finally, the last constraint is defined to maintain the original order of operations for every job in a given instance by counting the number of precedence violations between consecutive operations only.

$$\sum_{\substack{k_{n-1} < i < k_n \\ t + p_i > t'}} x_{i,t} x_{i+1,t'} \text{ for each } n$$

We name h_1, h_2, h_3 as the constraint objectives and we get

$$h_1(\bar{x}) = \sum_i \left(\sum_t x_{i,t} - 1 \right)^2$$

$$h_2(\bar{x}) = \sum_m \left(\sum_{(i,t,k,t') \in R_m} x_{i,t} x_{k,t'} \right)$$

$$h_3(\bar{x}) = \sum_n \left(\sum_{\substack{k_{n-1} < i < k_n \\ t + p_i > t'}} x_{i,t} x_{i+1,t'} \right)$$

where \bar{x} is a vector of length R representing all possible variables $x_{i,t}$. Note that if all three objectives have values equal to zero, then all constraints are met. In other words, there is a feasible schedule, and the makespan of this schedule is less than or equal to T . So, the resulting classical Hamiltonian can be expressed as $H_T(\bar{x}) = \eta h_1(\bar{x}) + \alpha h_2(\bar{x}) + \beta h_3(\bar{x})$ where the penalty constraints η , α and β must be larger than zero to ensure that unfeasible solutions do not have a lower energy than the ground states.

Thus far, the classical Hamiltonian for the Job Shop Scheduling Problem has been limited to finding a feasible schedule. However, it is crucial to acknowledge that JSP represents an optimization problem. We will add a penalty to optimize the makespan, favoring any optimal schedule over any non-optimal schedule. The additional objective is defined as

$$h_4(\bar{x}) = \sum_{n=1}^j (J+1)^{t_{i_n}}$$

and when the vector \bar{x} gives an optimal schedule its value becomes the lowest.

The concept behind this additional objective is based on the fact that only when a job's last operation is completed the job is considered finished. Therefore, the penalty can be applied only to the completion time of the final job's operations. The given penalty is of the form $base^{completion_t_of_last_op} = J+1^{completion_t_of_last_op}$.

Suppose that for our J jobs in a JSP instance, an optimal schedule finishes at time τ , while a non-optimal schedule finishes at a shorter time $\tau+1$. We denote with $o_{i_1}, o_{i_2}, \dots, o_{i_j}$ the last operations. Let $t_{i_1}, t_{i_2}, \dots, t_{i_j}$ be the last operations' completion times in an optimal schedule and $t'_{i_1}, t'_{i_2}, \dots, t'_{i_j}$ be the last operations' completion times of a non-optimal schedule. The penalty has the form of $\sum_{n=1}^j (J+1)^{t_{i_n}}$ with $t_{i_n} \leq \tau$ and it will be given to any optimal schedule. For the most penalized optimal schedule; $t_{i_n} = \tau$, i.e., the schedule in which all the last operations finish at time τ , the penalty has the form of $\sum_{n=1}^j (J+1)^\tau = J(J+1)^\tau$. We can see that

$$J(J+1)^\tau < (J+1)(J+1)^\tau = (J+1)^{\tau+1} \quad (1)$$

which means the non-optimal schedule is always more penalized than the optimal schedule with the most penalty.

It is only logical that any penalized optimal schedule will be smaller than most penalized optimal schedules, indeed

$$\sum_{n=1}^j (J+1)^{t_{i_n}} \leq J(J+1)^\tau \quad (2)$$

From (1) and (2) and for $t'_{i_n} = \tau+1$, we write $J(J+1)^\tau < (J+1)^{\tau+1} < \sum_{n=1}^j (J+1)^{t'_{i_n}}$

which proves that the most penalized optimal schedule is always less penalized than any non-optimal schedule.

Bibliography

- [1] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM Journal on Computing*, 37(1):166–194, 2007.
- [2] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1), January 2018.
- [3] Kostas Blekos, Dean Brand, Andrea Ceschini, Chiao-Hui Chou, Rui-Hao Li, Komal Pandya, and Alessandro Summer. A review on quantum approximate optimization algorithm and its variants. *Physics Reports*, 1068:1–66, June 2024.
- [4] Sergio Boixo, Vadim N. Smelyanskiy, Alireza Shabani, Sergei V. Isakov, Mark Dykman, Vasil S. Denchev, Mohammad Amin, Anatoly Smirnov, Masoud Mohseni, and Hartmut Neven. Computational role of collective tunneling in a quantum annealer, 2015.
- [5] Kelly Boothby, Paul Bunyk, Jack Raymond, and Aidan Roy. Next-generation topology of d-wave quantum processors, 2020.
- [6] Jacek Błażewicz, Erwin Pesch, and Małgorzata Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2):317–331, 2000.
- [7] Zhengjun Cao. A note on quantum approximate optimization algorithm. Cryptology ePrint Archive, Paper 2023/1854, 2023.

-
- [8] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.
- [9] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2):146–161, 1994. Project Management and Scheduling.
- [10] C. Carugno, M. Ferrari Dacrema, and P. Cremonesi. Evaluating the job shop scheduling problem on a d-wave quantum annealer. *Sci Rep*, 12:6539, 2022.
- [11] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, August 2021.
- [12] D-Wave Systems. What is quantum annealing? Accessed on: 2023.
- [13] D-Wave Systems. Programming the d-wave qpu: Setting the chain strength, April 2020. Accessed on: 2023.
- [14] Stéphane Dauzère-Pérès, Junwen Ding, Liji Shen, and Karim Tamssaouet. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2):409–432, 2024.
- [15] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.
- [16] Fred Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, 1989. 44
- [17] Fred Glover. Tabu search—part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990. 44
- [18] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models, 2019.

- [19] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000. 44
- [20] Siong Thye Goh, Jianyuan Bo, Sabrish Gopalakrishnan, and Hoong Chuin Lau. Techniques to enhance a qubo solver for permutation-based combinatorial optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 2223–2231, New York, NY, USA, 2022. Association for Computing Machinery.
- [21] David E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989. 44
- [22] Kazuki Ikeda, Yuma Nakamura, and Travis S. Humble. Application of quantum annealing to nurse scheduling problem. *Scientific Reports*, 9(1), September 2019.
- [23] A. K. Kaban, Z. Othman, and D. S. Rohmah. Comparison of dispatching rules in job-shop scheduling problem using simulation: A case study. *Int j simul model*, 11(3):129–140, 2012.
- [24] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 44
- [25] Wen-Yang Ku and J. Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016.
- [26] Krzysztof Kurowski, Tomasz Pecyna, Mateusz Słysz, Rafał Różycki, Grzegorz Waligóra, and Jan Węglarz. Application of quantum approximate optimization algorithm to job shop scheduling problem. *European Journal of Operational Research*, 310(2):518–528, 2023.
- [27] Krzysztof Kurowski, Jan Węglarz, Marek Subocz, Rafał Różycki, and Grzegorz Waligóra. Hybrid quantum annealing heuristic method for solving job shop scheduling problem. In Valeria V. Krzhizhanovskaya, Gábor Závodszy, Michael H. Lees,

- Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 502–515, Cham, 2020. Springer International Publishing.
- [28] Mark Lewis and Fred Glover. Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis, 2017.
- [29] Luc Libralesso, Vincent Jost, Khadija Hadj Salem, Florian Fontan, and Frédéric Maffray. Study on partial flexible job-shop scheduling problem under tooling constraints: complexity and related problems. In *European Journal of Operational Research*, 2019.
- [30] Matematija. Qubitrbm, 2022. Accessed: 2024-01-26. 74
- [31] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, Abhinav Kandala, Antonio Mezzacapo, Peter Müller, Walter Riess, Gian Salis, John Smolin, Ivano Tavernelli, and Kristan Temme. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, June 2018.
- [32] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- [33] Lotfi NOHAIR, Abderrahim EL ADRAOUI, and Abdelwahed NAMIR. Solving non-delay job-shop scheduling problems by a new matrix heuristic. *Procedia Computer Science*, 198:410–416, 2022. 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare.
- [34] G. Srinivisan NPTEL. Job shop scheduling - gantt chart, different dispatching rules, 2015. Accessed on: 2023.

-
- [35] Tobias Stollenwerk and Achim Basermann. Experiences with scheduling problems on adiabatic quantum computers. In *1st International Workshop on Post Moore's Era Supercomputing (PMES)*, 2016.
- [36] Benjamin C. B. Symons, David Galvin, Emre Sahin, Vassil Alexandrov, and Stefano Mensa. A practitioner's guide to quantum algorithms for optimisation problems, 2023.
- [37] Davide Venturelli, Dominic J. J. Marchand, and Galo Rojo. Quantum annealing implementation of job-shop scheduling, 2016.
- [38] P. Wojakowski and D. Warzolek. The classification of scheduling problems under production uncertainty. *Research in Logistics & Production*, 4(3):245–256, 2014.
- [39] Jiachen Zhang, Giovanni Lo Bianco, and J. Christopher Beck. Solving job-shop scheduling problems with qubo-based specialized hardware. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32(1):404–412, Jun. 2022.