Master's Thesis

Classical v/s Quantum Optimization: Comparison of Runtime Scaling

Submitted by:
Karthik Jayadevan
(Matriculation Number: 5582876)

Supervised by:
PD Dr. Thomas Wellens
Vanessa Dehn

Date of submission:
July 31, 2025

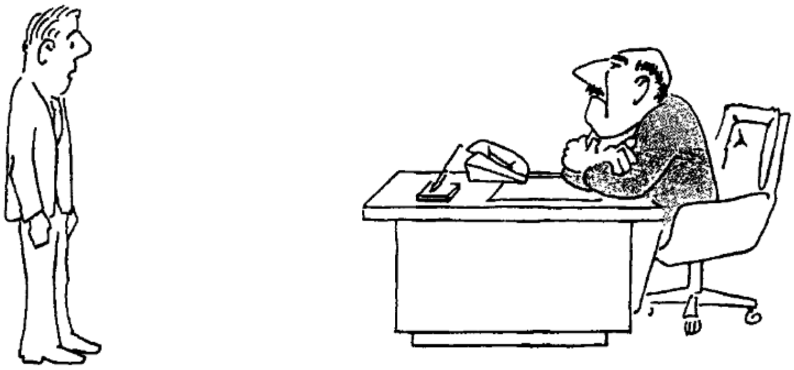
Albert-Ludwigs-Universität Freiburg
Fraunhofer Institut für Angewandte Festkörperphysik (IAF)

Abstract

The study of algorithms for combinatorial optimization has been a central focus of quantum computing research in recent years. The primary goal in this field is to develop efficient algorithms based on quantum mechanical principles, that can outperform (or more realistically, complement) classical algorithms for solving NP-hard problems. Setting up problem instances in the Quadratic Unconstrained Binary Optimization (QUBO) model, this thesis aims to compare the performance of classical algorithms like CPLEX, Goemans-Williamson and `MQLib` against two variants of the Linear Ramp Quantum Approximate Optimization Algorithm (LR-QAOA) on two combinatorial optimization problems: the Maximum Cut Problem (MaxCut) and a new resource allocation problem from the energy provider EnBW (which we refer to as the TA problem).

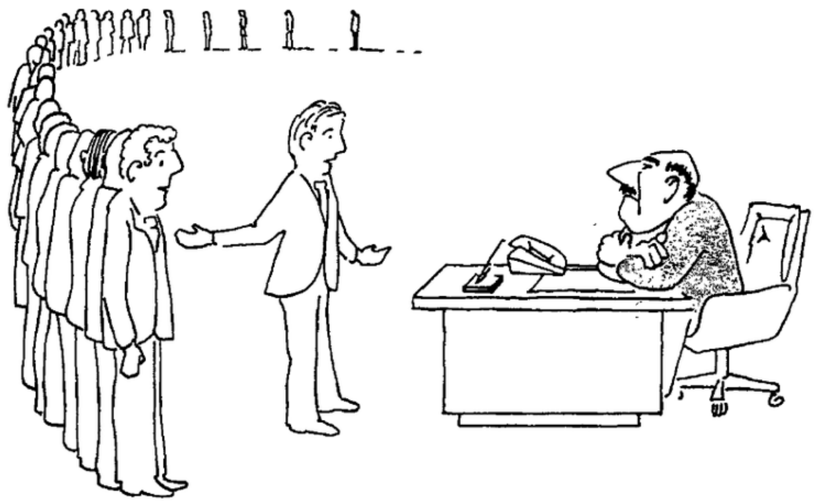
The first variant of LR-QAOA, which we refer to as normalized Hamiltonian LR-QAOA, sets up instance-agnostic parameters for the algorithm for all problem sizes, while the second variant, which we refer to as extrapolation LR-QAOA, determines suitable LR-QAOA parameters for each problem instance by heuristically extrapolating from smaller sub-instances.

For both the classical and quantum algorithms, we investigate how the runtime of each of the algorithms scales with the problem size for both problems, and how the solution quality compares across algorithms.



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, because no such algorithm is possible!"



"I can't find an efficient algorithm, but neither can all these famous people."

A comic strip from Reference [1] on finding efficient algorithms for optimization problems

Contents

Chapter 1

Introduction

Combinatorial optimization plays a pivotal role in a wide range of scientific and industrial applications [2]. These problems find use-cases in logistics [3], operations research [4] and theoretical computer science [5], among others. With the goal of finding an optimal solution from a finite yet exponentially large solution space, combinatorial optimization problems are often NP-hard [5, 6]. This implies that exactly solving instances of these problems becomes intractable for large problem sizes. This necessitates the need for approximation algorithms and heuristics that can provide near-optimal solutions even at large and complex problem instances.

State-of-the-art combinatorial optimization algorithms often rely on techniques such as branch-and-bound, semidefinite approximation, and heuristic methods [7]. Software libraries such as CPLEX [8] and MQLib [9] provide implementations of these algorithms, enabling users to solve combinatorial optimization problems efficiently. However, the complexity inherent in these problems often mean that the efficiency of these algorithms can come at the cost of solution quality, especially for large instances.

The recent surge in quantum computing research shows promise for optimization as a near-term application area, exploring the potential of quantum computers to solve combinatorial optimization problems more efficiently than classical computers [10, 11, 12]. While such an advantage over classical algorithms is yet to be demonstrated, hybrid approaches combining classical and quantum optimization routines emerge as a viable alternative. The Quantum Approximate Optimization Algorithm (QAOA) [13] is one of such hybrid algorithms that has been of prime interest in this domain [14]. This algorithm is characterized by its use of a parameterized quantum circuit, and a classical optimization routine to optimize the parameters $\vec{\gamma} = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ and $\vec{\beta} = \beta_1, \beta_2, \ldots, \beta_p$ of the circuit, over p iteration layers. The instance input to QAOA-based algorithms is provided as a Quadratic Unconstrained Binary Optimization (QUBO) problem, due to its structural similarity to the Ising Hamiltonian [15, 16]. Such a representation additionally enables a common representation for both classical and quantum optimization algorithms and allows for the use of existing classical optimization algorithms to solve the problem.

The classical parameter optimization routine in QAOA is found to be computationally hard [17, 18], and is often prone to local minima. An extension to QAOA, known as the linear ramp QAOA (LR-QAOA) has been recently proposed to address the limitations of the $\vec{\gamma}$ and $\vec{\beta}$ parameter optimization step [19, 20], where the parameters are scheduled using a linear ansatz, reducing the number of parameters to be optimized from $2p$ to just two, which are labelled Δ_γ and Δ_β. The approaches that have been proposed for implementing this algorithm are: (i) Normalization of the cost Hamiltonian and then using problem-agnostic Δ_γ and Δ_β values (i.e., constant for all problem instances) [19], or (ii) using an extrapolation based heuristic approach to optimize these parameters individually for each problem instance [20].

In this work, we compare the performance of both approaches to LR-QAOA against each other, as well as against efficient classical optimization algorithms such as CPLEX [8], Goemans-Williamson [21], and `MQLib` [9, 22]. The study is done by considering two types of QUBO problems: the Maximum Cut Problem (MaxCut) from graph theory, which is a well-known NP-hard combinatorial optimization problem [5], and a new industrial resource allocation problem (technician-asset allocation or TA problem) from the energy provider EnBW [23]. MaxCut is considered as a benchmark problem in quantum optimization, which makes it a suitable candidate for testing the LR-QAOA algorithm. The TA problem, whose cost function is derived from real data, helps us to evaluate the performance of LR-QAOA in a real-world scenario, and to compare it against classical optimization algorithms. The quantum algorithms are implemented using the Qiskit-Aer simulator [24], which gives us an opportunity to explore the idealized performance of these algorithms without the noise and errors present in real quantum hardware [12]. For both problems, across all algorithms studied, we investigate the question of how the time-to-solution scales with the problem size, also considering the optimality of the solution found. Importantly, the comparison is not made in terms of the absolute time taken to solve the problems, but rather in terms of how the runtime scales with increasing problem size, which is a more relevant metric for optimization algorithms.

Starting by establishing a conceptual foundation for combinatorial optimization and the problems considered in Chapter 2, the working principles of the classical optimization algorithms are discussed in Chapter 3. Then, Chapter 4 follows with an introduction to Quantum Optimization algorithms, starting with a brief introduction to the gate-based model of quantum computing. The chapter concludes after a description of the QAOA circuit, followed by a high-level overview of the LR-QAOA algorithm. The experiments set up to evaluate the runtime scaling of the algorithms are described in Chapter 5, where the analysis of classical algorithms is done over two size-scales: an asymptotic scaling for problem sizes N (up to 200 for MaxCut and 50 for TA) and a scaling analysis over a smaller problem range ($N \in [12, 28]$) so that the latter can be compared against the quantum algorithms. This chapter then describes the implementation of the two LR-QAOA approaches, and how the parameter optimization is performed in both cases. The runtimes of the quantum algorithms are quantified using a parameter which we refer to as the ‘total depth’ (defined as the depth of the circuit times the number of circuit executions required to find the optimal solution), which can be translated to a time measure by considering the gate execution times on specific quantum hardware. Finally, the chapter concludes with a direct comparison of the scaling exponents of the best classical algorithm against both LR-QAOA approaches. We summarize and discuss the results in Chapter 6, where we highlight the limitations of the study, and what direction future work can take to improve on the results presented in this work.

Chapter 2

Combinatorial Optimization and QUBO

Combinatorial Optimization is a branch of mathematical optimization that focuses on problems where the objective function is defined over a discrete set of solutions [6]. Before going into the specifics of combinatorial optimization, it is essential to understand the broader context of mathematical optimization, which serves as the foundation for combinatorial optimization. We further introduce the concept of Quadratic Unconstrained Binary Optimization (QUBO), which is a widely used framework for formulating combinatorial optimization problems.

Mathematical Optimization refers to the process of finding the best solution to a problem from a set of valid solutions. This process is crucial in various fields such as economics, engineering, logistics, and many others. The goal is to maximize or minimize a certain function that describes the problem which may include constraints that limit the possible solutions.

An optimization in a single variable involves a function $f(x)$ that we want to maximize or minimize, subject to constraints on the variable x. A minimization problem in one variable is mathematically written as:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & a \leq x \leq b, \end{aligned}$$

$a, b \in \mathbb{R}$ are the lower and upper bounds of the variable x, respectively. One can visualize the minimization process as finding the lowest point on a curve (and highest point on the curve for maximization), where the curve represents the function $f(x)$ and the constraints define the allowed range for x. Figure 2.1 illustrates this process for a single variable, showing how a function behaves within the constraints and where the minimum occurs.

2.1 Types of Optimization Problems

As described above, an optimization problem is generally characterized by an **objective function** (also called a cost function), which is to be maximized or minimized, subject to a set of constraints that define the feasible region of possible solutions. The value of the

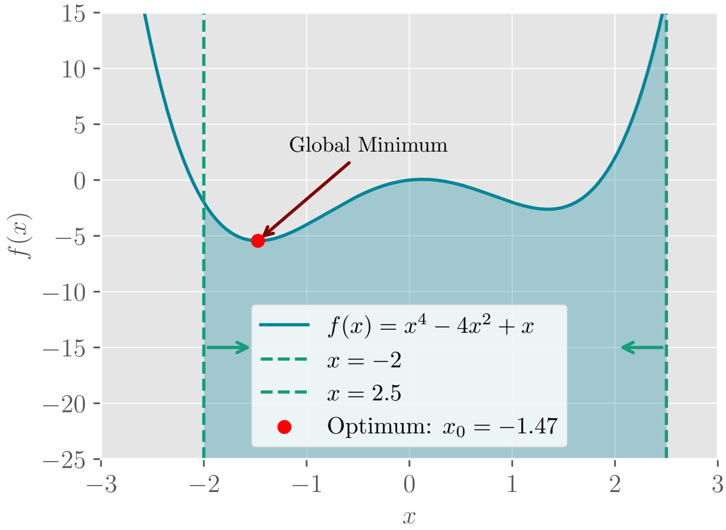


Figure 2.1: Example of minimization of the objective function $f(x) = x^4 - 4x^2 + x$ with constraints $-2 \leq x \leq 2.5$. The global minimum occurs at $x_0 \approx -1.47$, where $f(x_0) \approx -5.44$, and there exists a local minimum between $x = 1$ and $x = 2$. The shaded area represents the feasible region defined by the constraints.

objective function at a point $\mathbf{x}$ is often called the *cost* at that $\mathbf{x}$. A solution is considered *feasible* if it satisfies all the constraints, and the collection of all such solutions forms the solution space. The optimal solution is the feasible point that yields the best value (maximum value for a maximization problem and minimum value for a minimization) of the objective function.

For instance, in the example with a single variable (Figure 2.1), the objective function is $f(x) = x^4 - 4x^2 + x$, and the constraints are $-2 \leq x \leq 2.5$. The feasible region is the interval defined by these constraints, which is $[-2, 2.5]$. The optimal solution is the value of x within this interval that maximizes (or minimizes) the function $f(x)$.

In practice, optimization problems can be categorized into different types based on their characteristics. These include linear programming, integer programming, and semidefinite programming, among others [25, 7, 26]. Each type has its own set of methods and algorithms for finding solutions. Some classical optimizers which solve combinatorial optimization problems (see Chapter 3) convert the problem from one type to another, such as from integer programming to linear programming, to leverage the strengths of different optimization techniques. Hence it is useful to understand the definitions of the broad classes of optimization problems, as they provide a foundation for understanding the algorithms and techniques used to solve them.

Linear Programming

Optimization problems can be classified according to the nature of their objective functions and constraints. One fundamental class is Linear Programming (LP), where both the objective function and the constraints are linear. They are formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where c is a vector of coefficients for the objective function, A is a matrix representing the coefficients of the constraints, b is a vector representing the right-hand side of the constraints, and x is the vector of variables to be determined. The goal is to find the values of x that maximize (or minimize) the linear objective function while satisfying all linear constraints. The feasible region in LP forms a polyhedron, and the optimal solution is guaranteed to occur at one of its vertices.

Integer Programming

A more challenging class is Integer Programming (IP), in which some or all variables of an LP are restricted to integer values. An IP problem can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & x_i \in \mathbb{Z} \quad (i = 1, 2, \ldots, n) \end{aligned}$$

where $\mathbf{c}$ is a vector of coefficients for the objective function, A is a matrix representing the coefficients of the constraints, $\mathbf{b}$ is a vector representing the right-hand side of the constraints, and x_i are the integer variables. The integrality constraints on the variables make IP problems more complex than LP problems, as they can lead to non-convex feasible regions and multiple local optima. A comparison of the feasible region for an LP and an IP problem with the same objective and constraints is shown in Figure 2.2. The feasible region for the LP is a polyhedron, while the feasible region for the IP is a discrete set of points within that polyhedron.

There also exists a generalization of IP known as Mixed Integer Programming (MIP), where some variables are constrained to be integers while others can take any real value. MIP problems are particularly useful in practical applications and made use of in various solvers such as CPLEX [8] and Gurobi [27].

Typical solution methods for IP include branch-and-bound, cutting planes, and various heuristics.

A notable special case is Binary Integer Programming, where variables are restricted to 0 or 1, which is especially relevant in combinatorial optimization, which is introduced in Section 2.3.

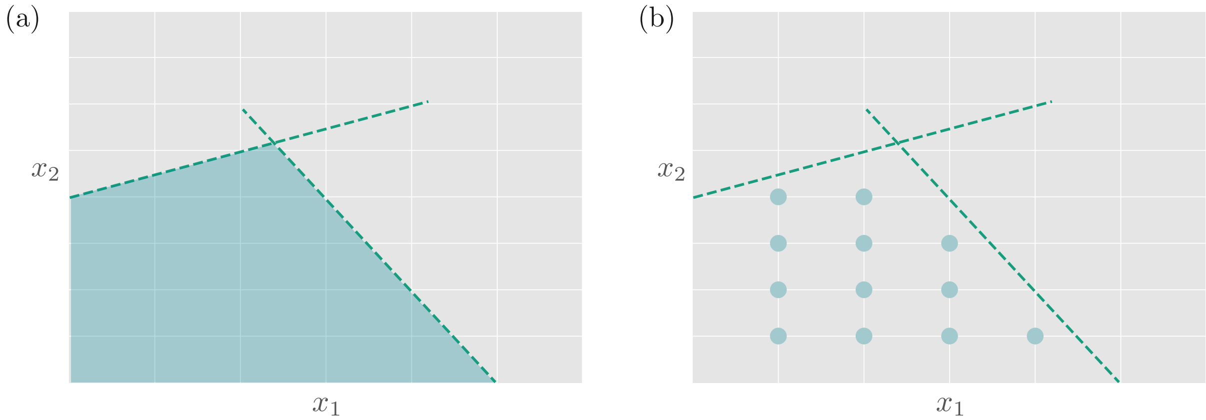


Figure 2.2: An example of feasible regions for (a) LP and (b) IP, for the same objective function (with variables x_1 and x_2) within the same polyhedron (bound by the constraints shown by the dashed lines). The feasible region for the LP is a continuous range of points, while the feasible region for the IP is a discrete set of points within that polyhedron.

Semi-Definite Programming

Semi-Definite Programming (SDP) further generalizes LP by allowing variables and constraints involving semi-definite matrices. A matrix is *semi-definite* if it is symmetric and all its eigenvalues are non-negative. SDP problems can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \langle C, X \rangle \\ \text{subject to} \quad & \langle A_i, X \rangle = b_i \quad (i = 1, 2, \ldots, m) \\ & X \succeq 0 \end{aligned}$$

where C and A_i are symmetric matrices, b_i are scalars, and $X \succeq 0$ is the positive semi-definite matrix variable to be determined. The notation $\langle A, B \rangle$ denotes the inner product of matrices A and B, defined as the sum of the products of their corresponding entries.

SDP is particularly valuable for problems with quadratic forms in the objective or constraints and is widely used in combinatorial optimization to obtain approximate solutions for NP-hard problems (which are problems of the complexity class NP, as introduced in Section 2.2.2). SDP relaxations can provide bounds on the optimal value, which are instrumental in the analysis and design of approximation algorithms.

2.2 Hardness of Problems

Many optimization problems can be efficiently solved by classical computers, but some problems are inherently difficult to solve. The difficulty need not imply that they cannot be solved, but rather that no known algorithm can solve them efficiently. The branch of computer science that studies the complexity of problems is known as computational complexity theory.

This discipline aims to prove lower bounds on the time and space required to solve problems, thereby establishing their inherent difficulty. In this section we define the key concepts in computational complexity theory, including Turing machines, complexity classes, further establishing the meaning of *efficiency* in the context of algorithms. Finally we introduce the concepts of heuristics and approximation algorithms, which are strategies used to tackle hard problems in optimization.

2.2.1 Turing Machines

The Turing machine is a theoretical model of computation that can simulate any algorithm. They consist of an infinite tape divided into cells, a head that reads and writes symbols on the tape, and a set of rules that dictate the machine's behavior based on the current state and the symbol being read.

A central concept in the theory of computation is the Church-Turing thesis. This thesis posits that any function which can be computed by an algorithm, regardless of the specific formalism or physical device, can also be computed by a Turing machine [28]. As a result, Turing machines are used as the standard model for defining what it means for a problem to be computable: any problem that cannot be solved by a Turing machine is considered uncomputable by any algorithmic means.

The *strong Church-Turing thesis* further asserts that any reasonable model of computation can be efficiently simulated by a probabilistic Turing machine, up to polynomial overhead. This means that, for the purposes of computational complexity theory, the Turing machine model is not only universal but also captures the efficiency of computation.

Consequently, computational complexity theory classifies algorithms according to how their running time scales with input size on a Turing machine. The two broad categories are:

- **Polynomial-time algorithms**: Algorithms whose running time is upper-bounded by a polynomial function of the input size (e.g., $O(n^k)$ for some constant k). These are considered "efficient" or "tractable".

- **Exponential-time algorithms**: Algorithms whose running time grows exponentially with input size (e.g., $O(2^n)$). These are considered "intractable" for large inputs.

This distinction is fundamental in complexity theory and underpins the classification of problems into complexity classes such as P, NP, NP-hard, and NP-complete, which are discussed in the next section.

2.2.2 Complexity Classes

Complexity classes are categories of problems based on their computational complexity. In this work, we will focus on four key complexity classes: P, NP, NP-hard, and NP-complete.

These classes help us understand the relationships between different problems and their solvability. In further discussions, when we refer to a problem as being "easy" to solve, we mean that there exists an algorithm that can solve the problem in 'polynomial time', i.e., the time taken to solve the problem grows polynomially with the size of the input. Conversely, when we refer to a problem as being "hard" to solve, we mean that no known algorithm can solve the problem in polynomial time.

Many optimization problems are associated with corresponding *decision problems*. A decision problem asks a yes/no question, such as whether there exists a feasible solution with an objective value at least (or at most) a given threshold. For example, instead of asking for the maximum number of items that can be packed in a knapsack (the Knapsack problem [1]), the decision version asks whether it is possible to pack items with total value at least V. Decision problems are central to complexity theory because complexity classes like P and NP are formally defined in terms of decision problems. This provides a standard framework to compare the relative hardness of various problems. Formally, complexity classes are categorized as follows:

1. **P**: Problems that can be solved in polynomial time by a deterministic Turing machine.
2. **NP**: Problems for which a solution can be verified in polynomial time by a deterministic Turing machine. This means that, given a solution to an NP problem, you can check whether it is correct in polynomial time. So while finding a solution may be hard, verifying a solution may not be.
3. **NP-hard**: Problems that are at least as hard as the hardest problems in NP. The important distinction is that NP-hard problems do not have to be in NP; they may not even have a solution that can be verified in polynomial time.
4. **NP-complete**: Problems that are both in NP and NP-hard. If any NP-complete problem can be solved in polynomial time, then all problems in NP can also be solved in polynomial time.

The relationship between these classes is a central question in computer science, particularly the famous P versus NP problem [1, 29], which asks whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time. The current consensus is that P is not equal to NP, but this has not been proven. Figure 2.3 illustrates the relationships between these complexity classes.

2.2.3 Algorithms: Heuristics versus Approximation

Since finding the optimal solution is a computationally hard task for many problems, especially those in NP-hard, various strategies have been developed to tackle these problems. The basic motive behind all of these approaches is to use algorithms that may not guarantee an optimal solution but can find a good enough solution in a reasonable amount of time. Such algorithms are broadly categorized into two types: heuristics and approximation

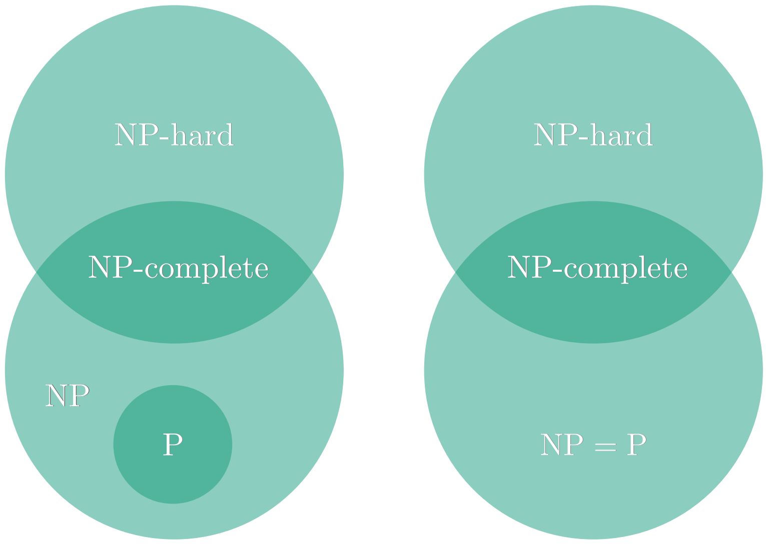


Figure 2.3: Complexity classes and their relationships. The figure illustrates the hierarchy of complexity classes, showing that P is a subset of NP, and NP-complete problems are the hardest problems in NP. NP-hard problems are at least as hard as NP-complete problems, but they may not be in NP themselves. The question of whether P equals NP remains an open problem in computer science.

algorithms.

The idea of heuristic algorithms is to provide a practical method to arrive at a "good" solution to a problem in polynomial time, even if that solution is not guaranteed to be the optimal one. They are approaches that do not provide any guarantees on how close the solution is to the optimal one. This means that while heuristics can often find good solutions quickly, it may not be trivial to determine how far the solution is from the optimal value.

A common example of a heuristic is the greedy algorithm, which, starting from an empty solution, iteratively adds the best available option at each step until no more options are available. This algorithm tends to often arrive at a local optimum, which may not be the global optimum, depending on the problem structure.

In contrast to heuristics, approximation algorithms, while also being polynomial-time algorithms that aim to find a near-optimal solution to an optimization problem, provide a guarantee on how close the solution is to the optimal one. An approximation algorithm is characterized by an *approximation ratio* [30], which is a measure of how close the solution is to the optimal solution. This quantity is the ratio of the cost (or value of the objective function) of the approximate solution to the cost of the optimal solution. We define the approximation ratio for a maximization problem as:

$$\alpha = \frac{\text{Cost of Approximate Solution}}{\text{Cost of Optimal Solution}} = \frac{f(x_{\text{approx.}})}{f(x_{\text{opt.}})}. \tag{2.1}$$

For a minimization, the ratio can be defined as the reciprocal of that for the maximization case, i.e.

$$\alpha = \frac{\text{Cost of Optimal Solution}}{\text{Cost of Approximate Solution}} = \frac{f(x_{\text{opt.}})}{f(x_{\text{approx.}})} \tag{2.2}$$

An algorithm with $\alpha = 0.7$ for an approximation algorithm, implies that the algorithm guarantees that the solution found will be at least 70% of the optimal solution. This is particularly useful in scenarios where finding the exact optimal solution is computationally infeasible, but a near-optimal solution is acceptable. An approximation algorithm with an approximation ratio of α is called an α-approximation algorithm.

2.3 Combinatorial Optimization

Combinatorial Optimization is the field of optimization where the objective function is defined over a discrete set of feasible solutions, i.e., the solution space is combinatorial in nature. In other words, these problems involve finding an optimal object from a finite set of objects [6].

Many Combinatorial Optimization Problems (or COPs) are NP-hard, with the solution space often increasing exponentially with the size of the input. Hence, they are often solved using heuristics or approximation algorithms, as finding the exact optimal solution is computationally infeasible for large problem instances.

Some of the most challenging COPs are listed in Karp's 21 NP-complete problems [31] and include problems such as the Traveling Salesman Problem, Knapsack Problem and Maximum Satisfiability Problem.

2.3.1 Quadratic Unconstrained Binary Optimization

COPs are commonly formulated using binary variables, where the solution is represented as a vector of binary values (0 or 1). A framework that is widely used to represent such problems is the Quadratic Unconstrained Binary Optimization (QUBO) model [32, 16]. In a QUBO problem, the objective function is expressed as a quadratic polynomial of binary variables, and the goal is to find a binary vector that minimizes (or maximizes) this polynomial.

The name "QUBO" stands for:

- **Quadratic**: The objective function is a quadratic polynomial.
- **Unconstrained**: The problem is formulated without *explicit* constraints, and any constraints are incorporated into the objective function through penalty terms (see [16] for examples of penalty formulation).
- **Binary**: The variables are binary, taking values in $\{0, 1\}$.
- **Optimization**: The goal is to find the binary vector that minimizes the objective

function $F(\mathbf{x})$.

$$\begin{aligned} \text{minimize} \quad F(\mathbf{x}) &= \mathbf{x}^T Q\, \mathbf{x} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} Q_{ij}\, x_i\, x_j \,. \end{aligned} \tag{2.3}$$

where $\mathbf{x}$ is a binary vector of length N, Q is a symmetric matrix of coefficients, with Q_{ij} representing the coefficient for the product $x_i x_j$, and $F(\mathbf{x})$ is the objective function to be minimized. The matrix Q is known as the *QUBO matrix*. Since the variables are binary, any linear terms in the objective function form the diagonal elements of the QUBO matrix, i.e., Q_{ii}, since $x_i^2 = x_i$ for binary variables.

In this work, we formulate all QUBO models as minimization problems, where the objective function is a quadratic polynomial of binary variables, and the goal is to find a binary vector that minimizes this polynomial. This has no loss of generality, as any maximization problem can be transformed into a minimization problem by negating the objective function, and vice versa. Some examples of problem formulation to QUBO are given in Appendix A.1. In this thesis, we will focus on the following problems that can be formulated as QUBO problems:

1. The Maximum Cut Problem (MaxCut) [31], a classical problem in graph theory and combinatorial optimization,

2. Technician-Asset allocation, a new industrial problem that involves optimizing the allocation of technicians and assets on a power grid (not to be confused with Reference [33] where the focus is on solving weighted MaxCut problems to determine optimal power-exchange sections, or Reference [34] where the objective is QUBO-based partitioning of the network for electricity-surplus redistribution).

2.3.2 The Maximum Cut Problem

Belonging to the field of graph theory, the Maximum Cut Problem (MaxCut) is a well-known NP-hard problem [31], which finds applications in circuit layout design and statistical physics [35]. In order to describe the problem, we first define some basic terminology related to graphs.

Graphs

A graph is a mathematical object defined using a set of *nodes* (or vertices) V and a set of edges E, where E consists of pairs of vertices that connect subsets of V. Graphs are usually denoted as $G(V, E)$. Graphs are visually represented with points denoting nodes and lines representing the edges between nodes, as shown in Figure 2.4(a). Note that the edge length in the visual representation has no significance.

According to the problem it represents, weights can be optionally assigned to the edges of a graph. An *unweighted* graph has the weights of all edges set to 1. Weighted graphs on

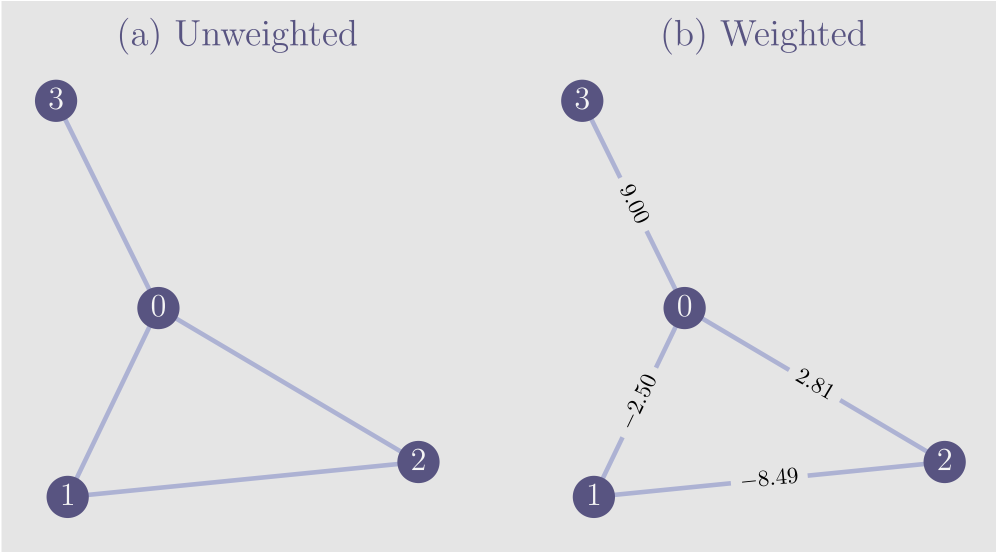


Figure 2.4: Graphs with 4 vertices and 4 edges ($V = \{0, 1, 2, 3\}$, $E = \{(0, 1), (0, 2), (0, 3), (1, 2)\}$): (a) unweighted, with weights of edges set to 1, (b) weighted, with weights assigned from a uniform distribution of range −10 to 10. This weighted graph has density 0.667 using Equation (2.5).

the other hand, can have any real values as weights to the edges.

To do mathematical computations on a graph, we use *adjacency matrices*, which are square matrices of order $N = \|V\|$ to represent the structure of the graph. For any graph, the adjacency matrix element for a pair of nodes i and j can be defined as[1]

$$A_{ij} = \begin{cases} w_{ij}, & i \text{ connected to } j \\ 0, & \text{else} \end{cases} \tag{2.4}$$

where w_{ij} is the weight of the edge (i, j). This implies the property that adjacency matrices are symmetric and have all diagonal entries zero. $w_{ij} = 1$ for an unweighted graph and $w_{ij} \in \mathbb{R}$ for weighted graphs.

Since each edge refers to a pair of nodes, a graph with N nodes can have up to $\binom{N}{2} = \frac{N(N-1)}{2}$ number of edges. The *density* of a graph is defined as the ratio of the number of edges $\|E\|$ present in the graph, to the total number of possible edges for the same number of nodes N, and reads as

$$\text{Density} = \frac{\|E\|}{\binom{N}{2}} = \frac{2\|E\|}{N(N-1)}. \tag{2.5}$$

The density takes value between 0 and 1. If the density of a graph is close to 1, they are referred to as *dense graphs*, and those with low density values are referred to as *sparse graphs*. However, these terms are more meaningful when used in particular context and in

[1]In this work, graphs with *loops* (where nodes can be connected to themselves) and those with edge-directions are not considered.

relation to other graphs in a data set.

Types of graphs

In addition to the broad categorization as weighted and unweighted, graphs can be further divided according to the nature of the number of edges in the graph. In this work the following graphs are used:

1. Regular graphs are those graphs in which all nodes have an equal number of edges in which they are part of. The number of edges a node has in a regular graph is called its degree d. In other words, a d-regular graph has each of its nodes connected to exactly d other nodes, and has $\frac{Nd}{2}$ number of edges.

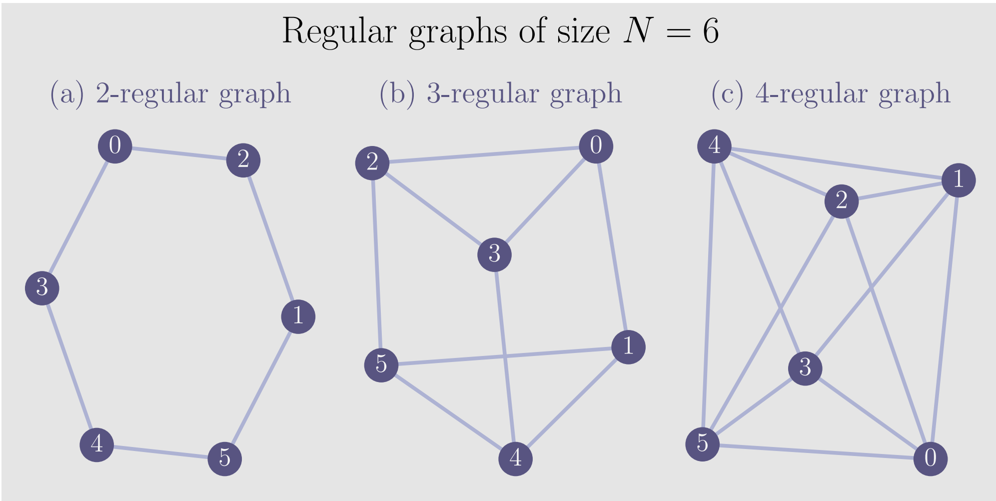


Figure 2.5: Regular graphs of varying degrees d for a fixed number of nodes N. These graphs are 'random' in the sense that the selection of nodes are to be connected is assigned randomly. Random-3-regular graphs are abbreviated as R3R graphs in this work.

   A valid regular graph should satisfy the conditions (i) $N \geq d + 1$ and (ii) the product $N \times d$ is even [36].

2. Complete graphs are fully connected graphs where every node is connected to every other node (see Figure 2.6). Hence these graphs have density of exactly 1. At any size N, there exists a unique complete graph.

3. Erdős-Rényi graphs (named after Paul Erdős and Alfréd Rényi who developed the algorithm to generate these graphs [37]) are random graphs where the probability of an edge between each pair of nodes is defined by a parameter p. Since they are characterized by the two inputs N and p, these graphs are also referred to as $G_{N,p}$ random graphs.

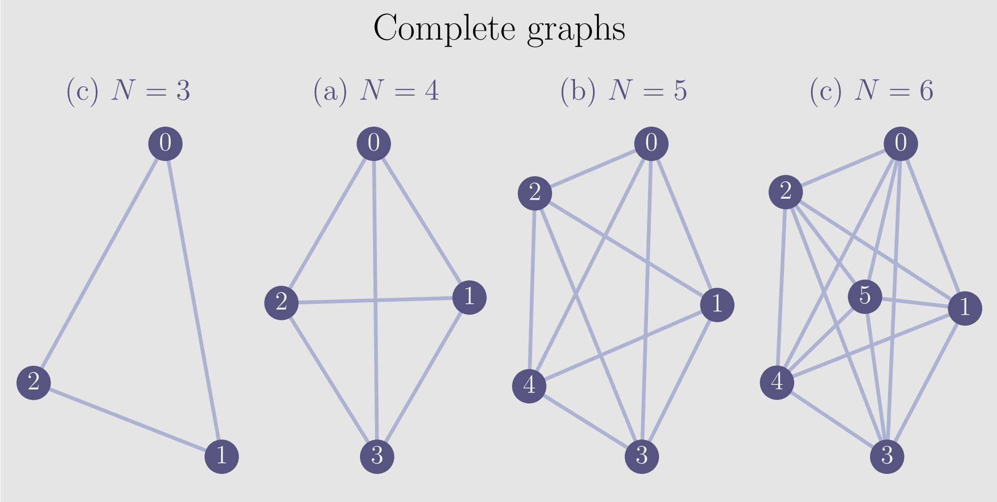


Figure 2.6: Complete graphs of increasing size N.

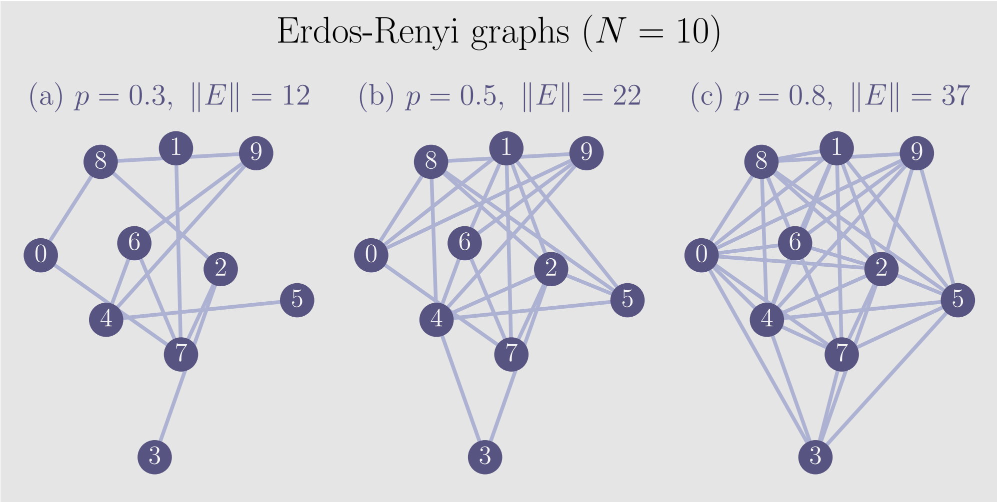


Figure 2.7: Erdős-Rényi graphs of increasing p, for the same set of nodes. It is easy to verify using Equation (2.5) that the density of these graphs are close to their respective p inputs that are used to generate them.

	Regular	Complete	Erdős-Rényi
Density	$\left(\dfrac{d}{N-1}\right)$	1	$\approx p$

Table 2.1: Comparison of densities of the three graph types.

The Maximum Cut Problem

The Maximum Cut Problem (more commonly known as MaxCut) is a well-known combinatorial optimization problem that involves partitioning the nodes of a graph into two disjoint sets such that the sum of the weights of the edges between the two sets is maximized. The objective function for the MaxCut problem for a graph $G(V, E)$ is formulated in V binary variables $x_i \in \{0, 1\}$, where $x_i = 1$ indicates that node i is in one set and $x_i = 0$ indicates that it is in the other set. The objective function can be expressed as:

$$\begin{aligned} \text{maximize} \quad & F(\mathbf{x}) = \sum_{(i,j)\in E} w_{ij}(x_i + x_j - 2x_i x_j) \\ & x_i \in \{0,1\},\ i \in \{1,\dots,V\} \end{aligned} \tag{2.6}$$

where w_{ij} is the weight of the edge between nodes i and j. The term $(x_i + x_j - 2x_i x_j)$ evaluates to 1 if nodes i and j are in different sets, and 0 if they are in the same set. Thus, the objective function sums the total weight of edges that connect two nodes in different sets.

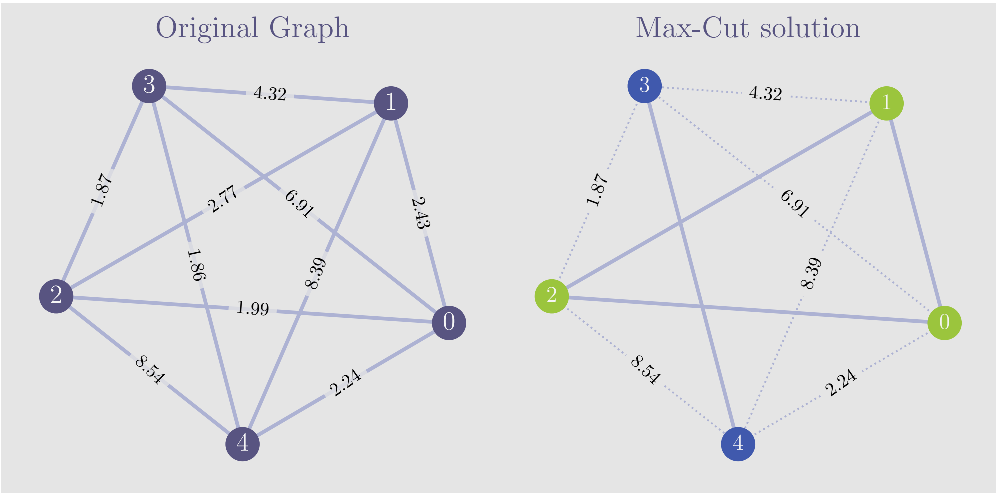


Figure 2.8: An example of the MaxCut problem with a complete graph of 5 nodes. The right side shows the optimal cut, which partitions the nodes into two sets such that the sum of the weights of the edges between the two sets is maximized. The dotted lines represent the edges that are cut by the partition, and the solid lines represent the edges that are not cut. The optimal cut in this case has a value of 32.27 (rounded to two decimal places), corresponding to the binary vector $\mathbf{x}_{\text{opt}} = (1, 1, 1, 0, 0)$.

The MaxCut solution for a graph of five nodes is illustrated in Figure 2.8. The landscape of the cost values for this graph is shown in Figure 2.9. The landscape plot shows that the optimal cost value is obtained at two indices. This is because the MaxCut problem is symmetric, meaning that the cut can be defined in either direction, and both cuts yield the same cost value. In other words, for any MaxCut problem, if $\mathbf{x}_{\text{opt}}$ is an optimal solution, then $\mathbf{1} - \mathbf{x}_{\text{opt}}$ is also an optimal solution, where $\mathbf{1}$ is a vector of all ones. This symmetry is reflected in the cost landscape, where the cost values are the same for both binary vectors.

The QUBO Formulation of MaxCut

The MaxCut problem translates naturally into the QUBO framework. The objective function in Equation (2.6) is already a quadratic polynomial in binary variables with no explicit constraints, making it suitable for the QUBO formulation. An instance of a MaxCut problem

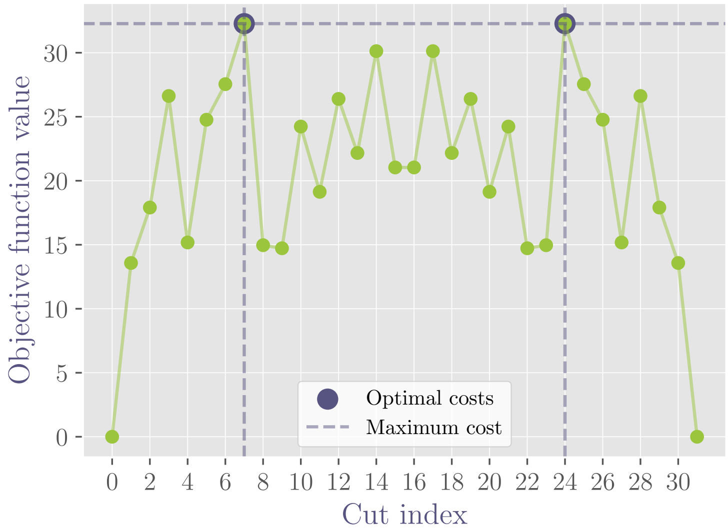


Figure 2.9: The cost landscape of the MaxCut problem for the example graph shown in Figure 2.8. The x-axis shows the integer representation of the binary vector $\mathbf{x}$. The y-axis shows the cost value of the objective function $F(\mathbf{x})$ for each binary vector. The optimal cost value is obtained at two indices, which correspond to the binary vectors $\mathbf{x}_{\text{opt}} = (1,1,1,0,0) \mapsto 24$ and $\mathbf{x}_{\text{opt}} = (0,0,0,1,1) \mapsto 7$, both of which yield the same cost value of approximately $F(\mathbf{x}_{\text{opt}}) = 32.27$ (rounded to two decimal places).

in N nodes can be formulated as a N-variable QUBO problem by defining the QUBO matrix Q as:

$$Q_{ij}^{(\text{MC})} = \begin{cases} w_{ij}, & i \neq j \\ -\sum_{k=1}^{N} w_{ik}, & i = j, \end{cases} \tag{2.7}$$

where w_{ij} is the weight of the edge between nodes i and j [38]. With this definition of the QUBO matrix, the MaxCut problem can be expressed as:

$$\begin{aligned} \text{minimize} \quad & F(\mathbf{x}) = \mathbf{x}^T Q^{(\text{MC})} \mathbf{x}, \\ & x_i \in \{0,1\},\ i \in \{1,\dots,N\}, \end{aligned} \tag{2.8}$$

where $Q^{(\text{MC})}$ is the QUBO matrix defined in Equation (2.7).

2.3.3 Technician-Asset allocation

The second COP we consider in this work is the industrial Technician-Asset allocation problem, proposed by the energy provider EnBW GmBH [23]. The company aims to optimize the allocation of their technicians to various assets in the power grid, such as transformers, substations, and power lines, distributed across the state of Baden-Württemberg in Germany.

We are given two sets of data: (i) a set of T technicians S_T, and (ii) a set of A assets S_A,

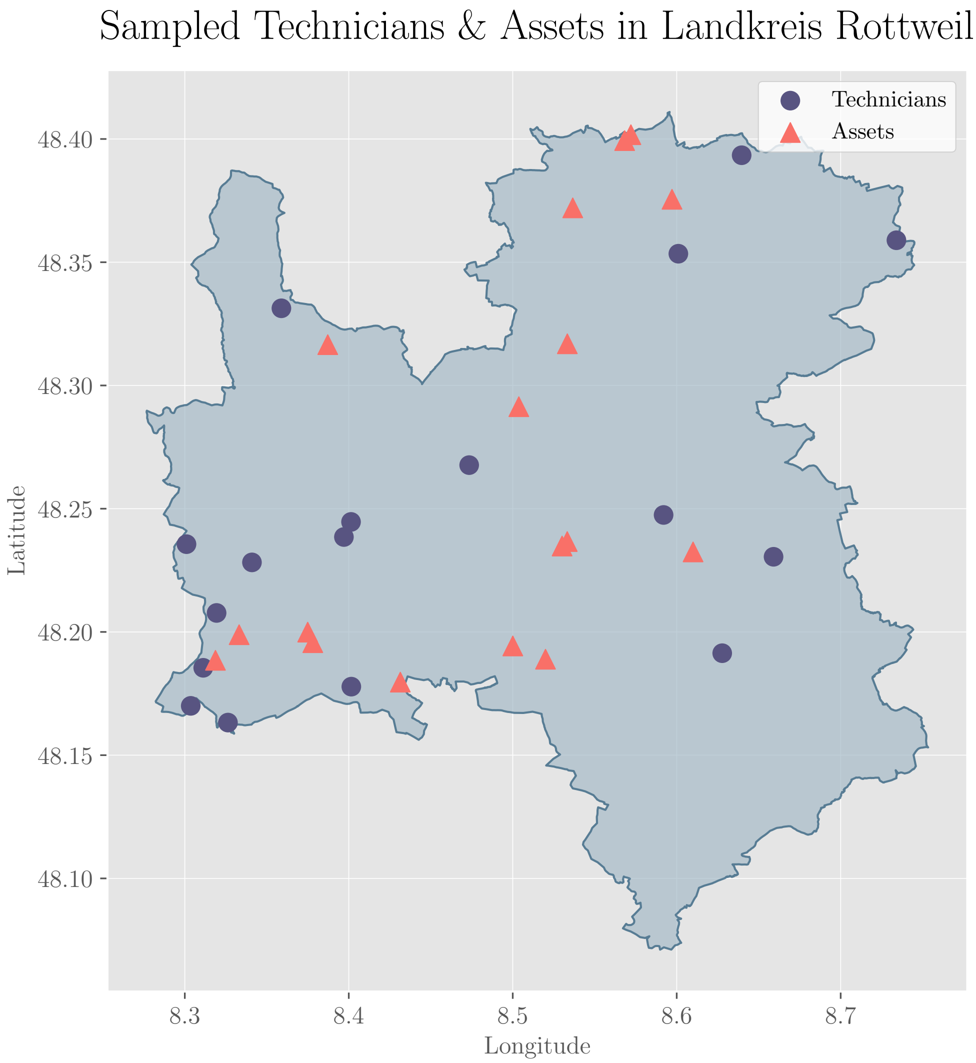


Figure 2.10: An example of the sampled technicians and assets in the Technician-Asset allocation problem for the district of Rottweil in Baden-Württemberg, Germany. The dots represent the technicians, and the triangles represent the assets.

both of which contain unique identifiers and specified geographic locations:

$$S_T = \{t_1, t_2, \ldots, t_T\}, \quad S_A = \{a_1, a_2, \ldots, a_A\}. \tag{2.9}$$

An example plot of the assets and technicians is shown in Figure 2.10, for the case of the district of Rottweil in Baden-Württemberg.

The technicians are responsible for maintaining the assets, and each technician can only be assigned to a subset of the assets. Each technician is also limited by the distance limit D_{lim} that they can travel to reach the assets. The goal of the problem (which we occasionally refer to as the TA problem in this work) is to partition the technicians and assets to groups such that the maximum distance traveled by any technician within a group is minimized, while ensuring that all assets are covered by at least one technician.

This problem, due to its novel and specific nature, has not been studied in the literature.

In order to have a meaningful QUBO formulation, we simplify this problem by considering the partition of technicians and assets into two groups, where each group has a set of technicians and a set of assets. This makes the formulation of the problem similar to the MaxCut problem, with binary variables representing the assignment of technicians and assets to the two groups. The binary variables are defined as follows:

$$x_{t,i} = \begin{cases} 0, & \text{technician } t_i \text{ assigned to set 0} \\ 1, & \text{technician } t_i \text{ assigned to set 1} \end{cases}, \quad x_{a,j} = \begin{cases} 0, & \text{asset } a_j \text{ assigned to set 0} \\ 1, & \text{asset } a_j \text{ assigned to set 1} \end{cases} \tag{2.10}$$

We further assert the condition that the set 1 contains exactly T_1 technicians and A_1 assets, and the set 0 contains exactly $T_0 = T - T_1$ technicians and $A_0 = A - A_1$ assets.

Cost Function

The cost function for the Technician-Asset allocation problem is defined as the average distance traveled by any technician within a group (to be minimized), plus a penalty term for each distance exceeding a given value D_{lim}. The cost function can be expressed as:

$$\tilde{F}(\mathbf{x}) = \sum_{i,j} W_{ij}^{(0)} \, (1 - x_{t,i}) \, (1 - x_{a,j}) + \sum_{i,j} W_{ij}^{(1)} \, x_{t,i} \, x_{a,j} \tag{2.11}$$

with $W_{ij}^{(0)}$ and $W_{ij}^{(1)}$ being the distance matrices for the two groups, which are defined as:

$$W_{ij}^{(0)} = \begin{cases} 2D_{\text{lim}} & \text{if } D_{ij} > D_{\text{lim}} \\ \frac{D_{ij}}{(A-A_1)(T-T_1)} & \text{if } D_{ij} \leq D_{\text{lim}} \end{cases}, \quad W_{ij}^{(1)} = \begin{cases} 2D_{\text{lim}} & \text{if } D_{ij} > D_{\text{lim}} \\ \frac{D_{ij}}{A_1 \, T_1} & \text{if } D_{ij} \leq D_{\text{lim}} \end{cases}, \tag{2.12}$$

where D_{ij} is the distance between technician t_i and asset a_j, and D_{lim} is the maximum distance that a technician can travel to reach an asset. The first term in Equation (2.11) corresponds to the cost for the technicians in set 0, and the second term corresponds to the cost for the technicians in set 1. If the distance between a technician and an asset exceeds the maximum distance limit D_{lim}, the cost is set to a large value $2D_{\text{lim}}$ to discourage such assignments. If the distance is within the limit, the cost is scaled by the number of technicians and assets in the respective sets to ensure that the cost is normalized, hence giving the average distance traveled by a technician in each group.

The choice of $2D_{\text{lim}}$ as the penalty for exceeding the distance limit originates from the fact that the average distance within one group is at most D_{lim}, and hence the cost function value in Equation (2.11) is at most $D_{\text{lim}} + D_{\text{lim}} = 2D_{\text{lim}}$ when all technicians are assigned to assets within the distance limit. Therefore, the cost function value is always at most $2D_{\text{lim}}$, and the penalty for exceeding the distance limit is set to this value so that solutions that violate the distance limit are discouraged.

Now we impose the added condition for the cardinality of the sets, which is that the set 1 contains exactly T_1 technicians and A_1 assets, and the set 0 contains exactly $T_0 = T - T_1$ technicians and $A_0 = A - A_1$ assets. This can be achieved by adding a penalty term to the cost function that increases the cost whenever the cardinality condition is violated. The penalty term can be expressed as:

$$P(\mathbf{x}) = \underbrace{2D_{\text{lim}}\left(T_1 - \sum_i x_{t,i}\right)^2}_{\text{penalty for } T} \quad + \quad \underbrace{2D_{\text{lim}}\left(A_1 - \sum_j x_{a,j}\right)^2}_{\text{penalty for } A}. \tag{2.13}$$

Bitstring assignments for $x_{t,i}$ and $x_{a,j}$ that satisfy the cardinality condition will have a Hamming weight (i.e., the number of 1's in the bitstring) of T_1 and A_1, respectively, and will therefore result in $P(\mathbf{x}) = 0$. If now the cardinality condition is not satisfied, the penalty term will penalize the objective function, thereby discouraging such solutions. The penalty is further scaled by a factor of $2D_{\text{lim}}$ to ensure that the penalty is significant enough to outweigh the cost of the objective function.

The final cost function for the Technician-Asset allocation problem can then be expressed as:

$$\text{minimize} \quad F(\mathbf{x}) = \tilde{F}(\mathbf{x}) + P(\mathbf{x}) \tag{2.14}$$

A critical point to note here is that a solution obtained is feasible only if it satisfies the constraint. The feasibility condition therefore dictates that

$$F(\mathbf{x}) < 2D_{\text{lim}} \tag{2.15}$$

for any valid solution. This is because the cost function value is at most $2D_{\text{lim}}$ when all technicians are assigned to assets within the distance limit and the penalty term evaluates to zero.

QUBO Formulation

The cost function Equation (2.14) is a quadratic polynomial in binary variables, with no explicit constraints, and can be expressed in the QUBO model. On expanding the cost function, we can express it as:

$$F(\mathbf{x}) = \text{constant} + \mathbf{x}^T Q^{(\text{TA})}\mathbf{x}$$

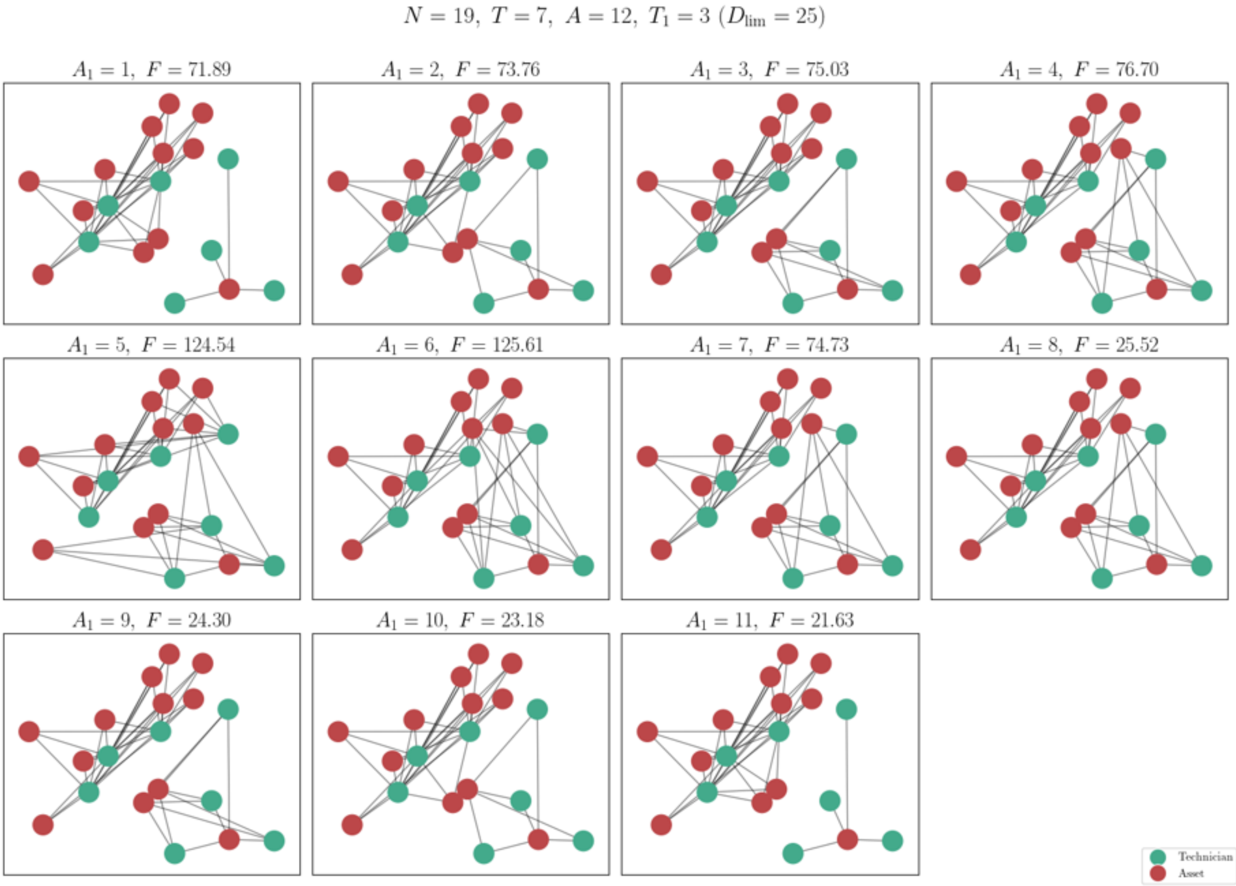


Figure 2.11: Examples of optimal solutions for the Technician-Asset allocation problem (shown as graph objects), with $T = 7$ technicians (green) and $A = 12$ assets (red). Technicians and assets belonging to the same set are connected. Each subplot shows the minimum value of the cost function evaluated for increasing A_1 (the number of assets in set 1) values for a fixed T_1 of $T_1 = 3$ and $N = 19$. The feasible solutions are the cases where $F < 50$, according to Equation (2.15).

with the QUBO matrix $Q^{(\mathrm{TA})}$ defined as:

$$Q^{(\mathrm{TA})}_{kl} = \begin{cases} 2D_{\mathrm{lim}} - \left[2T_1 \cdot (2D_{\mathrm{lim}}) + \sum_j W^{(1)}_{k,j}\right] \delta_{kl} & \text{if } k < T,\ l < T \\ \left(W^{(0)}_{k,l-T} + W^{(1)}_{k,l-T}\right)/2 & \text{if } k < T,\ l \geq T \\ \left(W^{(0)}_{l,k-T} + W^{(1)}_{l,k-T}\right)/2 & \text{if } k \geq T,\ l < T \\ 2D_{\mathrm{lim}} - \left[2A_1 \cdot (2D_{\mathrm{lim}}) + \sum_i W^{(1)}_{i,l-T}\right] \delta_{kl} & \text{if } k \geq T,\ l \geq T, \end{cases} \tag{2.16}$$

where δ_{kl} is the Kronecker delta function, which is 1 if $k = l$ and 0 otherwise. The constant term in this case is given by:

$$\text{constant} = 2D_{\mathrm{lim}}(T_1^2 + A_1^2) + \sum_{i,j} W^{(1)}_{ij}. \tag{2.17}$$

The resulting matrix is of size $N = T + A$. However, as mentioned in Section 2.3.1, the constant term can be ignored in the QUBO formulation, as it does not affect the optimization process. Therefore, the final QUBO formulation of the Technician-Asset allocation problem can be expressed as:

$$\begin{aligned} \text{minimize} \quad & F(\mathbf{x}) = \mathbf{x}^T Q^{(\mathrm{TA})} \mathbf{x} \\ & x_i \in \{0,1\},\ i \in \{1,\dots,T,\dots,T+A\}. \end{aligned} \tag{2.18}$$

The cost function evaluated for different choice of parameters is illustrated in Figure 2.11.

Chapter 3

Classical Algorithms for Combinatorial Optimization

For many combinatorial optimization problems, determining the optimal solution is challenging [1]. MaxCut being a well-known NP-complete problem, has been extensively studied in literature with a multitude of highly-tuned classical algorithms available for solving it: exact branch-and-cut algorithms enhanced by cutting plane techniques [39, 8], approximate algorithms using semi-definite programming (SDP) relaxations [21] and heuristic algorithms [38, 22, 40]. However, some of these methods are particularly tailored to the MaxCut problem, making use of its specific structure and properties. In contrast, quantum algorithms for combinatorial optimization problems are often designed to be more general [41, 13]. Hence, ensuring a fair comparison between classical and quantum algorithms requires a common problem representation. This work employs the Quadratic Unconstrained Binary Optimization (QUBO) formulation (see section Section 2.3.1), which is the native input for many quantum algorithms [16, 41], and is also widely used in classical optimization solvers. This representation guarantees that both the classical and quantum algorithms address the same problem objective over the same solution space, allowing for a more rigorous comparison of their performance.

The focus of this chapter is on solvers that are capable of solving general QUBO problems, allowing application to all the combinatorial optimization problems discussed in this work (Sections 2.3.1 to 2.3.3). More importantly, to enable the use of quantum algorithms (see Chapters 4 and 5) every problem instance is first mapped to a QUBO formulation before being processed by the algorithms.

Starting with the naïve brute-force search algorithm, which demonstrates the inherent difficulty of combinatorial optimization problems, this chapter further explores the CPLEX solver for exact solutions (Section 3.2), the Goemans-Williamson algorithm for approximate solutions using semi-definite programming relaxations (Section 3.3), and finally, heuristic solutions using MQLib (Section 3.4), an open-source package for the solution of QUBO problems using a variety of classical heuristic algorithms, including those designed for Max-

Cut. Each section provides insights into the strengths and limitations of these approaches, setting the stage for a comparison with quantum algorithms in subsequent chapters.

3.1 Brute-force

Any binary optimization problem can be solved by brute-force search, which involves evaluating all possible solutions and selecting the one that optimizes the objective function. For a given problem with cost function $F(\mathbf{x})$, where $\mathbf{x} \in \{0,1\}^N$, the brute-force approach can be described as given in Algorithm 3.1. The algorithm iterates through all 2^N possible binary vectors of length N, evaluates the cost function for each vector, and keeps track of the best solution found so far.

Algorithm 3.1: Brute-force search for binary optimization problems

Input: $F(\mathbf{x})$, where $\mathbf{x} \in \{0,1\}^N$
Output: $\mathbf{x}_0 = \arg\max_{\mathbf{x}} F(\mathbf{x})$
for $i = 0$ **to** $2^N - 1$ **do**
  $\mathbf{x} = \text{binary}(i, N)$;
  $f_i = F(\mathbf{x})$;
  if $f_i > f_{best}$ **then**
    $f_{\text{best}} = f_i$;
    $\mathbf{x}_0 = \mathbf{x}$;
return $\mathbf{x}_0$

This method is guaranteed to find the optimal solution, but its time complexity is $O(2^N)$, making it impractical for large N. For example, for $N = 10$, the algorithm has to evaluate more than 1000 solutions, for $N = 20$ over a million solutions, and for $N = 30$, it evaluates over a billion solutions. The exponential growth of the number of solutions evaluated for the MaxCut problem is illustrated in Figure 3.1.

For smaller problem sizes, brute-force search can be a useful tool to verify the correctness of other algorithms. Since brute-force search aims simply to evaluate all possible solutions, the scaling of the algorithm is independent of the problem. Note that we say *scaling* and not time-to-solve. The time-to-solve depends on the implementation and the nature of the cost function $F(\mathbf{x})$. However, the number of solutions evaluated is always 2^N, regardless of the problem, due to the binary nature of the optimization problem, making the rate of the time to solution scale as $O(2^N)$.

3.2 Exact solutions using Branch and Cut

Some combinatorial optimization problems can be solved by reducing them to a linear programming problem, which can then be solved using branch-and-cut algorithms. These algorithms are implemented in many commercial solvers, such as in IBM ILOG CPLEX Op-

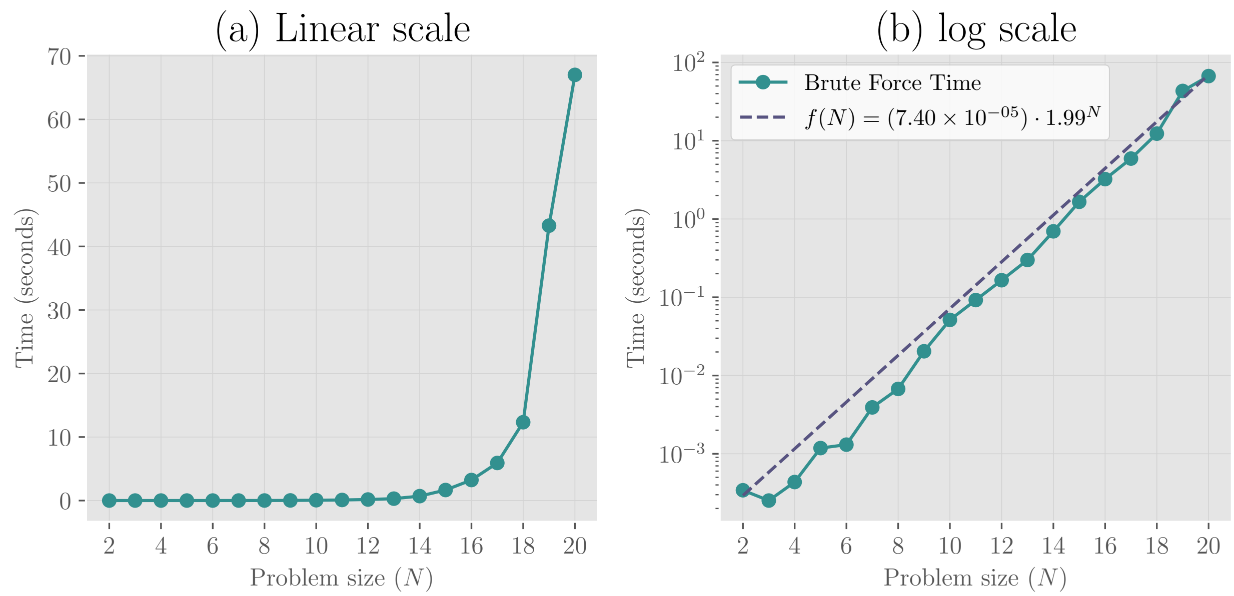


Figure 3.1: Scaling of brute-force search for MaxCut on a (a) linear and (b) log scale. (a) The number of solutions evaluated grows exponentially with the problem size N as 2^N. The plots show the time taken to evaluate all solutions for N ranging from 2 to 20, demonstrating the impracticality of brute-force search for larger problem sizes. The plot (b) shows the same data on a logarithmic scale, followed by fitting the data to $f(N) = a \cdot b^N$, where a and b are real constants. The fit (which becomes a straight line on a log scale) is shown as a dashed line, indicating the exponential growth of the number of solutions evaluated. At this scaling (on extrapolation), it would take about 31 hours to evaluate all solutions for $N = 30$ and over 32 years for $N = 40$.

timization Studio [8], commonly referred to as CPLEX. Named after the simplex method in linear programming (with an added "C" due to the implementation being in the C language), CPLEX is a powerful tool for solving large-scale linear programming, mixed-integer programming, and quadratic programming problems. It uses a combination of techniques including branch-and-bound, cutting planes, and heuristics to find optimal solutions efficiently.

Being a commercial solver, CPLEX is not open-source, but it is widely used in industry and academia for solving combinatorial optimization problems. Due to its proprietary nature, a description of the inner workings of the algorithm is limited. The general approach described in this section is based on the documentation [8, 42, 43] provided by the developers, complemented by studying the logs generated by the solver during the solution process (see Appendix B for an example).

The broad steps followed by the algorithm is illustrated as a flowchart in Figure 3.2. Once a QUBO instance is provided to CPLEX (see Section 5.1 for details), the solver initiates a multi-stage pipeline designed to efficiently find the optimal solution while maintaining global optimality guarantees. At high level, the algorithm (summarized in Algorithm 3.2) follows these steps:

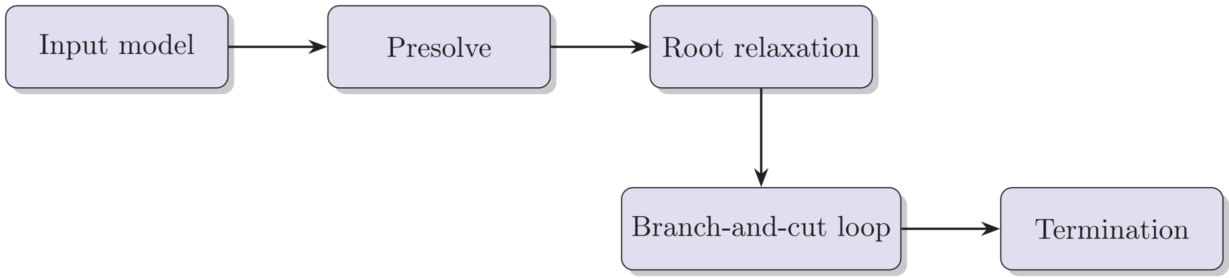


Figure 3.2: High-level overview of the CPLEX workflow for solving QUBO problems. The solver begins with presolve, where it simplifies the problem and decides whether to linearize the quadratic objective. It then solves the root relaxation to obtain bounds on the solution, followed by a branch-and-cut search that uses cuts, branching, and heuristics to explore the solution space. The process terminates when the optimality gap is within the specified tolerance.

1. **Presolve:** The solver begins with presolve, where it eliminates redundancies and tightens bounds. As part of presolve, CPLEX then decides whether to reformulate the quadratic objective into linear form, introducing auxiliary variables and constraints (and enabling its Mixed-Integer Linear Programming engine and cut-generation), or to retain the original objective and solve a Quadratic Program (QP) relaxation. In either case, the model emerging from presolve guarantees a valid dual bound at the root node (via an LP relaxation of the linearized form or a well-posed QP relaxation of the original). This strategic trade-off between model size and relaxation complexity is illustrated in the flowchart Figure 3.3.

2. **Root Node Relaxation:** The solver then solves the continuous relaxation of the problem. After presolve (and any linearization), this relaxation is either an LP (if the quadratic objective was linearized), or a QP (if the quadratic form was retained). In both cases the binary variables are relaxed to $[0, 1]$, and the relaxation provides a bound on the optimal objective.

3. **Branch-and-Cut Loop:** The core of the algorithm is the branch-and-cut loop, where the solver explores the solution space using a combination of branching, bound tightening, and cutting planes. The algorithm starts with the root node and iteratively branches on the binary variables to create subproblems. For each subproblem, it applies cutting planes to eliminate infeasible regions and tighten bounds on the optimal solution. The algorithm also employs heuristics to quickly find good feasible solutions, which can help in guiding the search.

4. **Termination:** The algorithm terminates when the optimality gap is within a specified tolerance, meaning that the difference between the best known feasible solution and the best bound on the optimal solution is small enough. At this point, the solver returns the best feasible solution found during the search. It is the feature of this

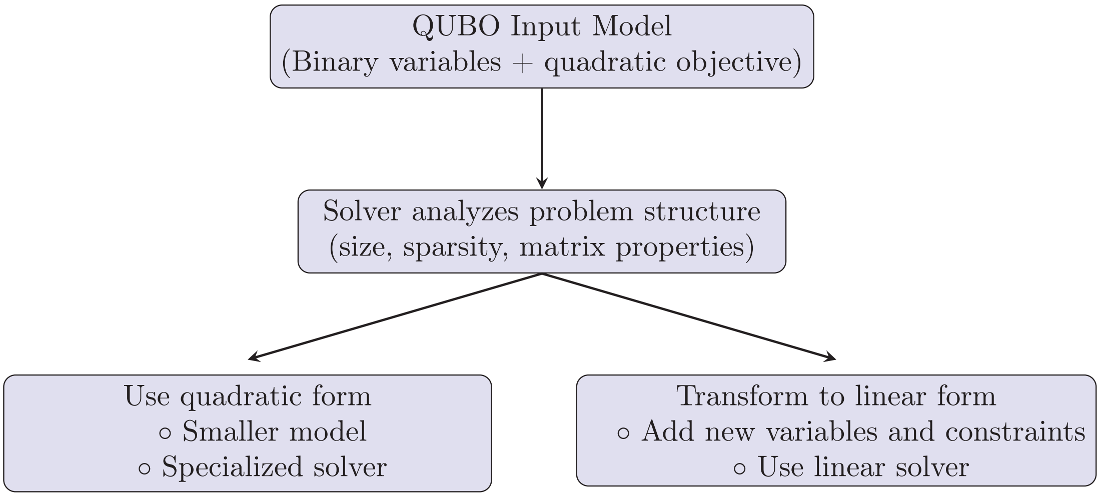


Figure 3.3: Reformulation paths taken by CPLEX (at the pre-solve step) when solving QUBO problems. Based on the problem's structure (such as the size, density, and definiteness of the quadratic matrix) the solver uses an internal decision model to either retain the quadratic form or transform the problem into a linear model with additional variables and constraints. This choice affects how the problem is processed internally and which optimization strategies are applied.

tolerance that allows CPLEX to find the optimal solution, by specifying a sufficiently small tolerance, *i.e.*, the algorithm can be run until it finds the optimal solution.

3.3 Approximate solutions using semi-definite relaxations

The Goemans-Williamson (GW) algorithm [21] is a well-known approximation algorithm for the MaxCut problem. It is an approximation algorithm (see Section 2.2.3) with an approximation ratio of 0.87856 (for undirected graphs with non-negative weights), meaning that the expected value of the cut found by the algorithm is at least 87.856% of the optimal cut value. The algorithm is based on semi-definite programming (SDP) relaxations and random hyperplane rounding. This was a significant milestone in the field of approximation algorithms, as it provided a polynomial-time algorithm that could guarantee a good approximation ratio (see Equation (2.1)) ; prior to this, the best proven approximation ratio was 0.5.

The algorithm includes the following steps:

1. Semi-definite programming relaxation
2. Solution extraction
3. Random hyperplane rounding

Unless otherwise specified, the following discussion is based on the original paper [21] with additional insights from References [44, 45] and [46].

Algorithm 3.2: CPLEX branch-and-cut algorithm for QUBO problems (high-level overview)

Input : QUBO instance defined over binary variables
Output: Best feasible solution found (incumbent)
Convert QUBO to internal model representation
  Apply presolve: Eliminate redundant constraints and variables
  Decide on quadratic-to-linear reformulation:
    if *linearized* **then**
      introduce auxiliary variables and linear constraints
    else
      keep original quadratic objective for QP relaxation
    end
  Tighten bounds; detect and exploit problem structure (e.g., fixed variables)
  Solve root relaxation: Relax binary constraints and solve LP (if linearized) or QP (if quadratic)
  Initialize incumbent if a feasible integer solution is found
Initialize branch-and-cut tree with root node
while *open nodes remain* ***and*** *optimality gap* $>$ *tolerance* **do**
  Select active node from tree (e.g., best-bound strategy)
  Apply cutting planes to strengthen the relaxation
  Solve node relaxation (LP or QP)
  if *solution is fractional* **then**
    Select branching variable and create child nodes
    Add child nodes to tree
  else
    if *solution improves incumbent* **then**
      Update incumbent
    end
  end
end
return incumbent solution

3.3.1 Semi-definite programming relaxation

As seen in Equation (2.6), the objective function for the MaxCut problem can be expressed as a quadratic form:

$$\textsc{MaxCut}: \quad \text{maximize} \qquad F(\mathbf{x}) = \sum_{(i,j)\in E} w_{ij}(x_i + x_j - 2x_i x_j),$$

where w_{ij} denotes the weight of the edge between nodes i and j, and E is the set of edges in the graph. The GW algorithm uses a modified version of this objective function, in terms of binary variables $z_i \in \{-1, 1\}$ instead of $x_i \in \{0, 1\}$ to represent the cut. This can be done using the relation $x_i = (1 - z_i)/2$. The objective function can be rewritten as:

$$\begin{aligned} \textsc{MaxCut}: \quad \text{maximize} \qquad F(\mathbf{z}) = \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - z_i z_j) \\ z_i \in \{-1, 1\} \quad \forall\, i, j \in V. \end{aligned} \tag{3.1}$$

It is easy to see that this formulation is equivalent to the original MaxCut problem, as the term in the bracket evaluates to 2 if nodes i and j are in different sets (cut) and 0 if they are in the same set (no cut). Now, the problem is first reformulated in terms of unit vectors $\mathbf{v}_i \in \mathbb{R}^n$ in place of the binary variables z_i. The unit vectors are defined such that $\|\mathbf{v}_i\|^2 = 1$ for all $i \in V$. The product term $z_i z_j$ is then replaced by the dot product $\mathbf{v}_i \cdot \mathbf{v}_j$. The objective function can thereby be expressed as:

$$\begin{aligned} \textsc{MaxCut}: \quad \text{maximize} \qquad F(\mathbf{v}) = \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - \mathbf{v}_i \cdot \mathbf{v}_j) \\ \|\mathbf{v}_i\|^2 = 1 \quad \forall\, i, j \in V. \\ \mathbf{v}_i = z_i \mathbf{v} = \pm\mathbf{v} \end{aligned} \tag{3.2}$$

This can be written using a matrix formulation as:

$$\begin{aligned} \textsc{MaxCut}: \quad \text{maximize} \qquad & F(Y) = \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - Y_{ij}) \\ \text{subject to:} \qquad & Y_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j \\ \forall\, i, j \in V, \qquad & \mathbf{v}_i = z_i \mathbf{v},\ z_i = \pm 1 \\ & \|\mathbf{v}_i\| = 1 (\implies Y_{ii} = 1) \end{aligned} \tag{3.3}$$

where $Y \in \mathbb{R}^{n\times n}$ is a matrix variable with $Y_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. Note that the matrix Y is a symmetric rank-1 positive semidefinite matrix [46]. The next steps of the GW algorithm involves relaxing the rank-1 restriction on the matrix Y to allow it to be any positive semidefinite

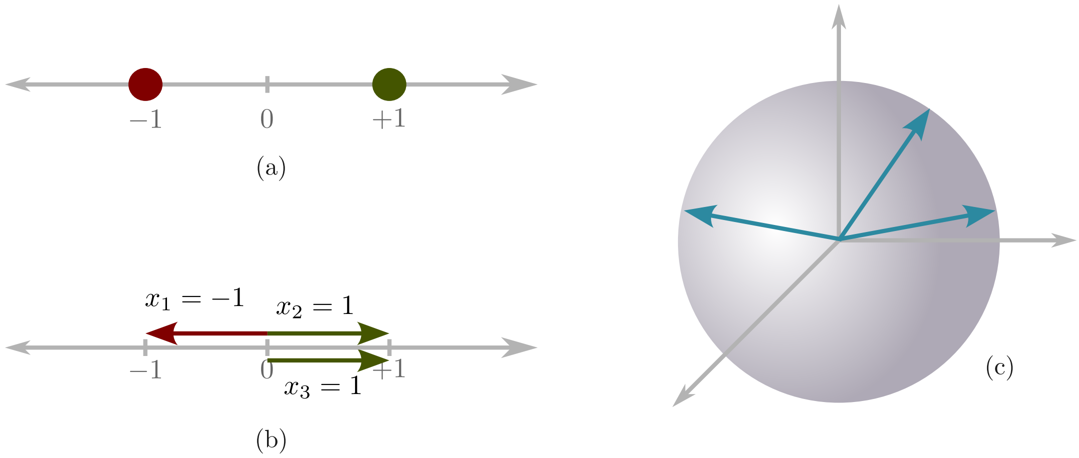


Figure 3.4: Illustration of the relaxation from binary variables to unit vectors in the Goemans-Williamson algorithm. (a) The original binary variables $z_i \in \{-1, 1\}$ represent the cut, where $z_i = 1$ indicates that vertex i is in one set of the cut and $z_i = -1$ indicates that it is in the other set. (b) Representation in terms of unit vectors $\mathbf{v}_i \in \mathbb{R}^2$, mapping $z = -1$ to $\mathbf{v} = (-1, 0)$ and $z = 1$ to $\mathbf{v} = (1, 0)$. (c) Relaxation of the binary variables to unit vectors in $\mathbb{R}^n$, where the binary values are replaced by unit vectors on the unit sphere as in Equation (3.4).

matrix, while still retaining the diagonal entries to be one. This relaxation is written as:

$$\begin{aligned} \textsc{Relax:} \quad & \text{maximize} & F(Y) = \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - Y) \\ & \text{subject to:} & Y_{ii} = 1, \quad Y \succeq 0 \\ & & \forall\, i, j \in V. \end{aligned} \tag{3.4}$$

An important remark to note is that the matrix Y of the relaxed problem is no longer directly related to the variables corresponding to the nodes (z_i). This information has to be now retrieved from the matrix Y indirectly. The advantage of this relaxation is that now the problem has been boiled down to solving the SDP in Equation (3.4), which is proven to be efficiently solvable [21]. Therefore, by the end of this step, we have a matrix $Y^* \in \mathbb{R}^{n \times n}$ that is the solution to the SDP relaxation of the MaxCut problem (see Chapter 5 for details on the solver used).

3.3.2 Solution extraction: random hyperplane rounding

Now that we have the solution to the SDP relaxation, the next step is to extract the binary solution from the matrix Y^*. In the version of the problem given in Equation (3.3), the binary value of the variable z_i indicated whether the vertex i is in the cut or not. However, in the relaxed problem, the matrix Y^* does not directly provide this information. So the next step is to extract the unit vectors $\mathbf{v}_i$ from the matrix Y^*. This is done by performing a decomposition technique like eigen decomposition or Cholesky decomposition [47] of the matrix Y^*, which gives us a matrix $V \in \mathbb{R}^{n\times n}$ such that

$$Y^* = V^\top V, \tag{3.5}$$

where $V = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$. The vectors obtained, while still not binary, are unit vectors in $\mathbb{R}^n$, which can be interpreted as points on the unit sphere in n-dimensional space.

The final step of the GW algorithm is to convert the unit vectors $\mathbf{v}_i$ into binary values $z_i \in \{-1, 1\}$, which indicate whether the vertex i is in the cut or not. This is done using a *random hyperplane rounding technique*:

1. Sample a random unit vector $\mathbf{r} \in \mathbb{R}^n$ from a standard normal distribution. This can be visualized as choosing a random hyperplane (plane in n-dimensional space).

2. Take the dot product of the random vector $\mathbf{r}$ with each unit vector $\mathbf{v}_i$ to obtain a scalar value $s_i = \mathbf{r}^\top \mathbf{v}_i$. This dot product gives a measure of how aligned the unit vector $\mathbf{v}_i$ is with the random vector $\mathbf{r}$. In other words, it determines on which side of the hyperplane the point $\mathbf{v}_i$ lies (see Figure 3.5).

3. Assign the binary value z_i based on the sign of s_i:

$$z_i = \begin{cases} 1 & \text{if } s_i \geq 0, \\ -1 & \text{if } s_i < 0, \end{cases} \tag{3.6}$$

   i.e., if the dot product is non-negative, assign $z_i = 1$, otherwise assign $z_i = -1$.

4. The final partition is then given by the set of vertices with $z_i = 1$ and those with $z_i = -1$.

The complete algorithm is summarized in Algorithm 3.3. The random hyperplane rounding step is crucial as it ensures that the expected value of the cut found by the algorithm is at least 87.856% [21] of the optimal cut value, making it a powerful approximation algorithm for MaxCut.

3.4 Heuristic solutions

Finally, we discuss heuristic solutions using the open-source package `MQLib` [38, 9]. An acronym for **M**axCut **Q**UBO **Lib**rary, `MQLib` is a versatile library of heuristic algorithms designed to solve MaxCut and general QUBO problems using a variety of classical algorithms. It provides a unified interface for different solvers, allowing users to easily switch between them and compare their performance on the same problem instance.

Algorithm 3.3: Goemans–Williamson algorithm for MaxCut using SDP relaxation

Input : Undirected graph $G = (V, E)$ with edge weights $w_{ij} \geq 0$
Output: Binary vector $\mathbf{z} \in \{-1, 1\}^n$ representing a cut

SDP relaxation

Formulate and solve the SDP:

$$\max_{Y \succeq 0,\, Y_{ii}=1} \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - Y_{ij})$$

Let Y^* be the optimal solution matrix

Extract unit vectors from Y^*

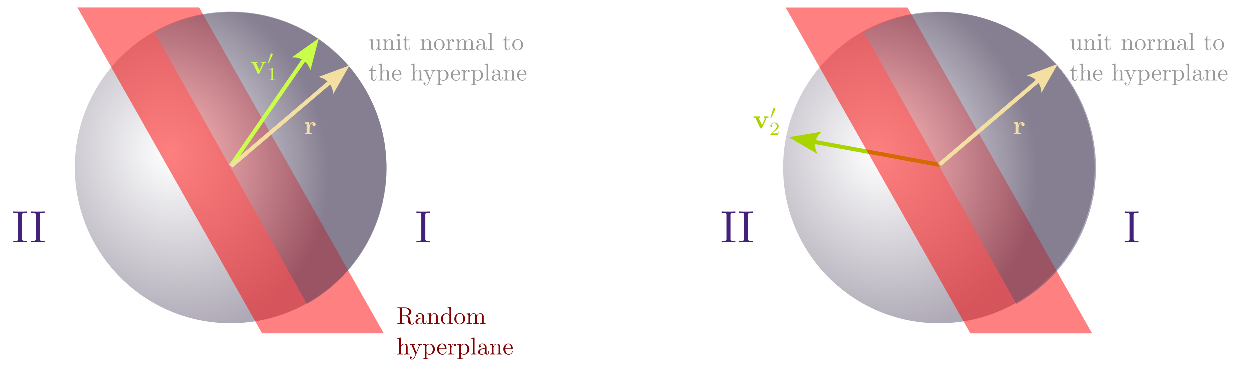
Perform Cholesky or eigen-decomposition:

$$Y^* = V^\top V, \quad V = [\mathbf{v}_1, \ldots, \mathbf{v}_n].$$

Each $\mathbf{v}_i \in \mathbb{R}^n$ is a unit vector

Random hyperplane rounding

Sample a random unit vector $\mathbf{r} \in \mathbb{R}^n$
foreach $i \in V$ **do**
  Compute $s_i = \mathbf{r}^\top \mathbf{v}_i$
  if $s_i \geq 0$ **then**
    $z_i \leftarrow 1$
  else
    $z_i \leftarrow -1$
  end
end
return $\mathbf{z} = (z_1, \ldots, z_n)$



(a) (b)

Figure 3.5: Illustration of the random hyperplane rounding step in the Goemans-Williamson algorithm (in three dimensions). The figure shows a unit sphere with points representing the unit vectors $\mathbf{v}_i$ (green arrow), a random plane (red quadrilateral) defined by the random vector $\mathbf{r}$ (yellow arrow). The resulting binary partition is based on the sign of the dot product $s_i = \mathbf{r}^\top \mathbf{v}_i$. This can be visualized as projecting the vectors $\mathbf{v}_i$ onto the vector $\mathbf{r}$ and checking whether the projection is positive or negative. In other words, vectors which are on the same side of the hyperplane are assigned to the same set of the cut, while those on the other side are assigned to the other set. Each side of the hyperplane is denoted by I and II, where the vectors in side I are assigned $z_i = 1$ and those in side II are assigned $z_i = -1$. (a) The vector $\mathbf{v_1'}$ gets assigned $z_1 = 1$ as it is on side I of the hyperplane, while (b) the vector $\mathbf{v_2'}$ gets assigned $z_2 = -1$ as it is on side II of the hyperplane.

In this work we primarily focus on the heuristic `BURER2002` which follows the rank-two relaxation heuristic approach described in [22]. This heuristic builds upon the Goemans-Williamson algorithm, to find a good feasible solution for large MaxCut instances. As shown in Section 3.3, the GW algorithm uses an N-dimensional relaxation of the SDP, which is then rounded using random hyperplane rounding to obtain a binary solution. However, this algorithm does not scale well for large problems. For example, a problem with $N = 1000$ would require solving a 1000×1000 SDP (i.e., 10^6 variables), which is computationally expensive. In other words, the GW algorithm does not scale well for large problems due to the high dimensionality of the SDP relaxation [22].

The Burer-Monteiro approach reduces this problem to a rank-two relaxation. This avoids the need for a full N-dimensional SDP (and associated $O(N^2)$ variables of the SDP), and instead uses a rank-two matrix $Y \in \mathbb{R}^2$ to represent the solution. However, this comes at the cost of losing the optimality guarantees of the GW algorithm. The Burer-Monteiro algorithm is a heuristic that provides a good feasible solution for large MaxCut instances, but it does not provide an approximation guarantee as the GW algorithm does.

3.4.1 The Burer-Monteiro algorithm

The algorithm starts by replacing the binary variables $z_i \in \{-1, 1\}$ by unit vectors $\mathbf{v}_i \in \mathbb{R}^2$ (in contrast to the R^N vectors used in the GW algorithm). The set of N unit vectors $\mathbf{v}_i$ is mapped as a single vector $\Theta = (\theta_1, \ldots, \theta_N)^\top \in \mathbb{R}^N$, called the *angular representation of a cut*:

$$\mathbf{v}_i = \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix}, \quad i = 1, \ldots, N, \tag{3.7}$$

which then transforms the MaxCut objective function into a function of the angles θ_i (written as a minimization problem):

$$\begin{aligned} \text{minimize} \quad & F(\Theta) = \frac{1}{2} \sum_{(i,j)\in E} w_{ij} \cos(\theta_i - \theta_j) \\ & \forall\ i, j \in V. \end{aligned} \tag{3.8}$$

Note that the dot product $\mathbf{v}_i \cdot \mathbf{v}_j$ has been replaced by the cosine of the angle between the two vectors, which is equivalent to the original formulation.

This formulation allows the gradient to be computed efficiently, further enabling the use of the gradient descent method to find the optimal angles θ_i, establishing that every cut $\tilde{\Theta}$ corresponds to a stationary point of the objective function $F(\Theta)$. The authors further prove that only the maximum cut corresponds to a local minimum of $F(\Theta)$.

The angular representation of the cut further offers a natural way to map the angles to binary values, where the cut is defined as:

$$\theta_i = \begin{cases} 0, & \text{if } z_i = 1, \\ \pi, & \text{if } z_i = -1, \end{cases} \tag{3.9}$$

which is analogous to the random hyperplane rounding step shown in Section 3.3.2.

This algorithm, on specifying an upper bound on the runtime, will keep perturbing the angles θ_i to get better feasible solutions, and repeat the optimization routine until the time limit is reached.

Chapter 4

Quantum Optimization

The field of quantum computing involves the study of the application of quantum mechanical principles to perform certain computations. This domain has seen significant advancements in recent years, particularly in the context of optimization problems [11]. Quantum optimization leverages quantum algorithms to find optimal or near-optimal solutions to specific problems, with the potential to outperform classical approaches [12].

Quantum algorithms for combinatorial optimization problems have been a focal point of research, with notable algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) [13] and its variants [14].This chapter provides an overview of the fundamental concepts in quantum computation, focusing on the quantum circuit model. Further sections delve into specific quantum optimization techniques, including the QAOA and its variant the Linear Ramp QAOA [19, 20].

4.1 Fundamentals of Quantum Computation

The fundamentals of quantum computation are built upon the principles of quantum mechanics, which govern the behavior of quantum systems. This section introduces key concepts such as qubits, the Bloch sphere representation, and the quantum circuit model, which are essential for understanding how quantum algorithms operate. Unless otherwise specified, the discussion in this section is based on the References [28, 48].

Classical information is represented using bits, which can take values of either 0 or 1. In contrast, quantum information is represented using a quantum state vector in a two-dimensional Hilbert space, known as a *qubit*. The commonly used orthonormal basis states of a qubit are denoted in the Dirac notation as $|0\rangle$ and $|1\rangle$, analogous to the classical bits. These basis states are referred to as the *computational basis states*. Any arbitrary state of a qubit can be expressed as a linear combination of these basis states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{4.1}$$

where $\alpha, \beta \in \mathbb{C}$ and satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. The state $|\psi\rangle$ is said to be a superposition of the basis states $|0\rangle$ and $|1\rangle$. The coefficients α and β represent the probability amplitudes of measuring the qubit in the states $|0\rangle$ and $|1\rangle$, respectively. Equation (4.1) can equivalently be expressed in matrix form as:

$$|\psi\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{4.2}$$

where we defined the basis states as column vectors $|0\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ and $|1\rangle = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$.

The state of a single qubit can be visualized geometrically on the Bloch sphere. In this representation, the state of a qubit is represented as a point on the surface of a unit sphere, where the north and south poles correspond to the states $|0\rangle$ and $|1\rangle$, respectively. The polar and azimuthal angles (θ and ϕ respectively) on the Bloch sphere define the state of the qubit (up to a global phase):

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle, \tag{4.3}$$

where $\theta, \phi \in \mathbb{R}$. In the Bloch sphere representation, superposition states are represented by points on the surface of the sphere. This is shown in Figure 4.1.

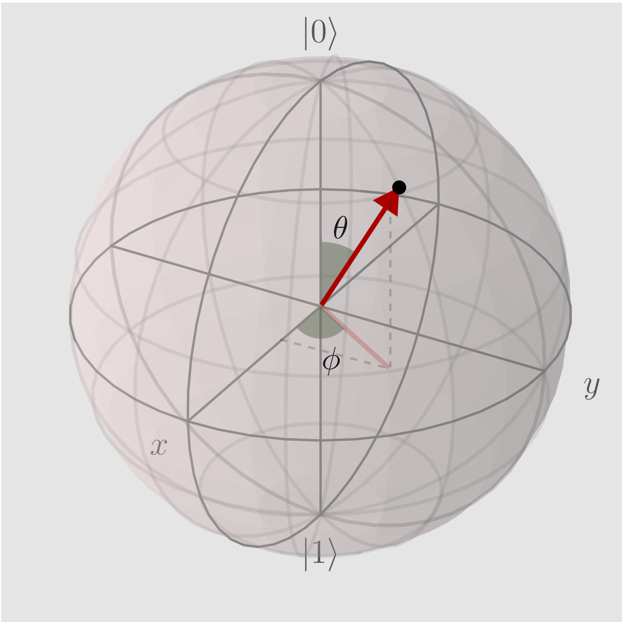


Figure 4.1: The Bloch sphere representation of a qubit. The state of a qubit can be visualized as a point on the surface of the sphere, where the north pole represents the state $|0\rangle$ and the south pole represents the state $|1\rangle$. Superposition states are represented by points on the surface, with angles θ and ϕ defining their position. Points inside the sphere represent mixed states, which are probabilistic mixtures of pure states (points on the sphere) rather than coherent superpositions.

While the representation of information in a qubit allows for an infinite number of states by an arbitrary choice of α and β in Equation (4.1), the measurement of a qubit collapses its state to one of the basis states, either $|0\rangle$ or $|1\rangle$. The probability of obtaining the state $|0\rangle$

upon measurement in the computational basis is given by the square of the absolute value of the amplitude α, and similarly for $|1\rangle$ with amplitude β. This is expressed mathematically as:

$$P(|0\rangle) = \| \langle 0|\psi\rangle \|^2 = \|\alpha\|^2, \quad P(|1\rangle) = \| \langle 1|\psi\rangle \|^2 = \|\beta\|^2. \tag{4.4}$$

The state of a quantum system consisting of multiple qubits is described by a tensor product of the individual qubit states. The tensor product of two qubit states $|\psi_1\rangle$ and $|\psi_2\rangle$ is written as $|\psi_1\rangle \otimes |\psi_2\rangle$, often denoted simply as $|\psi_1\psi_2\rangle$. For n qubits, the state can be expressed as a linear combination of the computational basis states, which are represented as $|b_1b_2\dots b_n\rangle$, where each $b_i \in \{0,1\}$. The general form of a multi-qubit state of n qubits, in a Hilbert space of dimension 2^n is given by:

$$|\psi\rangle = \sum_{b_1,b_2,\dots,b_n \in \{0,1\}} c_{b_1b_2\dots b_n} |b_1b_2\dots b_n\rangle, \tag{4.5}$$

where $c_{b_1b_2\dots b_n}$ are complex coefficients representing the probability amplitudes of each basis state $|b_1b_2\dots b_n\rangle$. The n-qubit Hilbert space is the tensor product of each one-qubit Hilbert space: $\mathcal{H}_n = \mathcal{H}_1 \otimes \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_1$. The normalization condition for multi-qubit states is given by:

$$\sum_{b_1,b_2,\dots,b_n \in \{0,1\}} |c_{b_1b_2\dots b_n}|^2 = 1. \tag{4.6}$$

For example, a two-qubit state can be expressed as:

$$|\psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle, \tag{4.7}$$

where the Hilbert's space is spanned by $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, and c_{ij} are complex coefficients representing the probability amplitudes of each basis state, satisfying the normalization condition:

$$|c_{00}|^2 + |c_{01}|^2 + |c_{10}|^2 + |c_{11}|^2 = 1. \tag{4.8}$$

The Quantum Circuit Model

The quantum circuit model provides a framework for representing quantum computations by a sequence of discrete, unitary operators called *gates* acting on a set of qubits. A quantum circuit diagram (see Figures 4.2 and 4.3) is used to visually represent such a computation. The diagram is read from left to right: each horizontal line ("wire") represents one qubit, and symbols placed on those wires denote gates applied at successive steps. Quantum gates implement unitary operators U satisfying $U^\dagger U = I$ and admit an inverse $U^\dagger$.

Formally, a circuit built from gates $U_1, U_2, \dots, U_m$ implements the overall unitary

$$U_{\text{circuit}} = U_m U_{m-1} \cdots U_2 U_1,$$

where the rightmost gate U_1 is applied first to the input state. In the Schrödinger picture, if $|\psi_0\rangle$ is the initial state, then the unitary evolution of the quantum state by the circuit is given by

$$|\psi_{\text{out}}\rangle \;=\; U_{\text{circuit}}\,|\psi_0\rangle\,.$$

Because each gate is a $2^k \times 2^k$ unitary (for a k–qubit operation), the entire circuit acts as a $2^n \times 2^n$ unitary on an n-qubit register. The construction of the quantum circuit for the QAOA involves both single-qubit and multi-qubit gates, which are applied in a specific sequence to prepare the quantum state that encodes the solution to the optimization problem. These types of gates are defined below.

Single-Qubit Gates

A single qubit gate is a unitary operation that acts on a single qubit, represented by a 2×2 unitary matrix U. The action of a single qubit gate on a qubit state $|\psi\rangle$ is given by:

$$|\psi'\rangle = U\,|\psi\rangle\,, \tag{4.9}$$

where U is the unitary matrix representing the gate, and $|\psi'\rangle$ is the resulting state after the gate is applied. Common single qubit gates include the Pauli gates, Hadamard gate, and phase gates.

1. **Pauli Gates:** The Pauli gates are defined using the Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{4.10}$$

The Pauli X gate is also known as the NOT gate because it flips the state of a qubit, i.e.,

$$\begin{aligned} X\,|0\rangle &= |1\rangle\,, \\ X\,|1\rangle &= |0\rangle\,, \end{aligned} \tag{4.11}$$

just like the classical NOT gate. The Pauli Z gate has no effect on the state $|0\rangle$ and introduces a phase shift of π to the state $|1\rangle$:

$$\begin{aligned} Z\,|0\rangle &= |0\rangle\,, \\ Z\,|1\rangle &= -\,|1\rangle\,. \end{aligned} \tag{4.12}$$

The Pauli Y gate maps the states $|0\rangle$ and $|1\rangle$ as:

$$\begin{aligned} Y\,|0\rangle &= i\,|1\rangle\,, \\ Y\,|1\rangle &= -i\,|0\rangle\,. \end{aligned} \tag{4.13}$$

2. **Hadamard Gate:** The Hadamard gate is a single qubit gate that maps the compu-

tational basis states to the equal superposition states:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{4.14}$$

The Hadamard gate transforms the computational basis states into the $|+\rangle$ and $|-\rangle$ basis states defined by:

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \equiv |+\rangle, \\ H|1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \equiv |-\rangle. \end{aligned} \tag{4.15}$$

Using Equation (4.4), we can see that the probability of measuring both $|0\rangle$ and $|1\rangle$ for a qubit in the state $|+\rangle$ is equal to $\frac{1}{2}$. This is also the case for the state $|-\rangle$, which is why these states are often referred to as equal superposition states. The Hadamard gate is particularly useful for creating such superposition states, which are essential for quantum algorithms that rely on interference effects.

3. **Rotation Gates:** Rotation gates are single qubit gates that perform rotations around the x, y, or z axes of the Bloch sphere. The rotation gates are defined as:

$$\begin{aligned} R_x(\theta) \equiv e^{-i\frac{\theta}{2}X} &= \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \\ R_y(\theta) \equiv e^{-i\frac{\theta}{2}Y} &= \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \\ R_z(\theta) \equiv e^{-i\frac{\theta}{2}Z} &= \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}. \end{aligned} \tag{4.16}$$

The Pauli Z gate in Equation (4.12) is equivalent to $R_z(\pi)$, up to a global phase factor. This can be seen by substituting $\theta = \pi$ into the definition of the R_z gate:

$$R_z(\pi) = e^{-i\frac{\pi}{2}Z} = \begin{pmatrix} e^{-i\frac{\pi}{2}} & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = -i \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = -iZ.$$

All of the single qubit gates defined above are unitary operations, meaning they preserve the norm of the quantum state. They can be visualized on the Bloch sphere, where the action of these gates corresponds to rotations and reflections of the qubit state vector on the surface of the sphere.

The application of quantum gates can be represented as a sequence of unitary operations applied to the qubit state. For example, applying a Hadamard gate H followed by a rotation gate $R_x(\theta)$ to a qubit state $|\psi\rangle$ can be expressed as:

$$|\psi'\rangle = R_x(\theta)H|\psi\rangle. \tag{4.17}$$

A key fact to note here is that, in the circuit notation of Equation (4.17) the Hadamard gate is drawn first, followed by the rotation gate, as described at the beginning of the section.

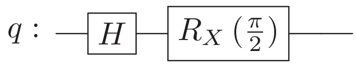


Figure 4.2: Application of single qubit gates. The figure illustrates the application of a Hadamard gate followed by a rotation gate $R_x(\pi/2)$ to a qubit state $|q\rangle$, represented as a horizontal wire. While the circuit notation shows the Hadamard gate followed by the rotation gate (as the circuit is read from left to right), when writing down the matrix representation, the unitary operation is $R_x(\pi/2)H\,|q\rangle$, indicating that the Hadamard gate is applied first, followed by the rotation gate.

Multi-Qubit Gates

Multi-qubit gates are unitary operations that act on two or more qubits simultaneously. These gates are essential for creating entanglement between qubits, which is a key resource in quantum computing, with no classical analogue. Entangled states are multi-qubit states which cannot be written as a tensor product of single qubit states. For example, the Greenberger-Horne-Zeilinger (GHZ) state

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$$

is an entangled state of three qubits, as it cannot be expressed as a tensor product of single qubit states. In the context of QAOA, multi-qubit gates are essential to encode the interactions between the variables in the QUBO problem.

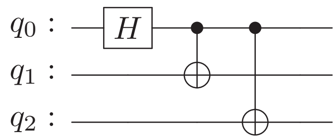


Figure 4.3: Quantum circuit representation of single and multi-qubit gates. The figure shows a quantum circuit with three qubits, where the Hadamard gate is applied to the first qubit (q_0), followed by two CNOT gates that entangle q_0 (control qubit) with qubits q_1 and q_2 (target qubits). The resulting state is the Greenberger-Horne-Zeilinger (GHZ) state $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$.

1. **CNOT Gate:** The Controlled-NOT (CNOT) gate is a two-qubit gate that flips the state of the target qubit if the control qubit is in the state $|1\rangle$. The CNOT gate is

represented by the following unitary matrix:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.18}$$

The action of the CNOT gate on the computational basis states is given by:

$$\begin{aligned} CNOT\,|00\rangle &= |00\rangle\,, \\ CNOT\,|01\rangle &= |01\rangle\,, \\ CNOT\,|10\rangle &= |11\rangle\,, \\ CNOT\,|11\rangle &= |10\rangle\,. \end{aligned} \tag{4.19}$$

When applied to superposition states, the CNOT gate may create entanglement between the control and target qubits. The circuit representation of the CNOT gate is shown in Figure 4.3.

2. **CZ Gate:** The Controlled-Z (CZ) gate is another two-qubit gate that applies a Pauli-Z operation to the target qubit if the control qubit is in the state $|1\rangle$. The CZ gate is represented by the following unitary matrix:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{4.20}$$

The action of the CZ gate on the computational basis states is given by:

$$\begin{aligned} CZ\,|00\rangle &= |00\rangle\,, \\ CZ\,|01\rangle &= |01\rangle\,, \\ CZ\,|10\rangle &= |10\rangle\,, \\ CZ\,|11\rangle &= -\,|11\rangle\,. \end{aligned} \tag{4.21}$$

3. R_{zz} **Gate:** The R_{zz} gate is a two-qubit gate defined as:

$$R_{zz}(\theta) = \begin{pmatrix} R_z(\theta) & 0 \\ 0 & R_z(-\theta) \end{pmatrix} \tag{4.22}$$

where $R_z(\theta)$ is the single-qubit rotation gate defined in Equation (4.16). Thus, the R_{zz} matrix is a 4×4 unitary matrix that acts on two qubits. The R_{zz} operation can be expressed in terms of the Pauli-Z gate as:

$$R_{zz}(\theta) = e^{-i\frac{\theta}{2}Z\otimes Z} = e^{-i\frac{\theta}{2}Z_1Z_2}, \tag{4.23}$$

where Z_1 and Z_2 are the Pauli-Z operators acting on the first and second qubits, respectively. The parameter θ represents an angle of rotation. The action of the R_{zz} gate on the computational basis states is given by:

$$\begin{aligned} R_{zz}(\theta)\,|00\rangle &= e^{-i\frac{\theta}{2}}\,|00\rangle\,, \\ R_{zz}(\theta)\,|01\rangle &= e^{i\frac{\theta}{2}}\,|01\rangle\,, \\ R_{zz}(\theta)\,|10\rangle &= e^{i\frac{\theta}{2}}\,|10\rangle\,, \\ R_{zz}(\theta)\,|11\rangle &= e^{-i\frac{\theta}{2}}\,|11\rangle\,. \end{aligned} \tag{4.24}$$

From this equation, we can see that the R_{zz} gate is symmetric with respect to the states $|01\rangle$ and $|10\rangle$, which means that the 'control' and 'target' qubits are interchangeable. This is why the R_{zz} gate is often represented with both qubits as control qubits, as shown in Figure 4.4. A useful result for the R_{zz} gate is that it can be expressed in terms of the CNOT gate and single qubit gates. The R_{zz} gate is particularly useful for implementing the cost Hamiltonian in the QAOA algorithm (see Section 4.2.3), as it allows for the encoding of interactions between qubits.

(a) (b)

Figure 4.4: (a) Circuit notation of the R_{zz} gate. Due to its symmetric nature, the R_{zz} gate can be represented with both qubits as control qubits. (b) The R_{zz} gate can be decomposed into a CNOT gate, an R_z gate, and another CNOT gate. This decomposition is useful for implementing the R_{zz} gate in quantum circuits.

Having introduced the basic model of quantum computation that we will use in this work, we now turn our attention to understanding how optimization problems can be formulated to be solved using quantum algorithms.

4.2 Quantum Optimization Algorithms

Quantum optimization algorithms aim to solve optimization problems by encoding the cost function into a Hamiltonian acting on qubits, and then executing unitary operations to prepare the ground state of the Hamiltonian (or its approximation). In this work, we focus on quantum algorithms based on the quantum circuit model. This section outlines how optimization problems can be formulated to be solved using quantum algorithms, starting with the method of encoding classical optimization problems into quantum circuits. We then introduce the Quantum Approximate Optimization Algorithm (QAOA) and its variant, the Linear Ramp QAOA (LR-QAOA).

As discussed in Section 2.1, optimization problems can be formulated as finding the minimum (or maximum) of a cost function $f(x)$ over a set of feasible solutions. We also saw that combinatorial optimization problems can be expressed as QUBO problems (Section 2.3.1). Quantum optimization algorithms typically encode these problems into quantum circuits by representing the cost function as a Hamiltonian H, which is a mathematical operator that describes the energy of a quantum system. For an eigenstate $|\psi\rangle$ of the Hamiltonian H,

$$H |\psi\rangle = E |\psi\rangle \,, \tag{4.25}$$

where E is the energy eigenvalue corresponding to the eigenstate $|\psi\rangle$. In other words, the energy of a physical system in its eigenstate is given by the expectation value of the Hamiltonian:

$$E = \langle H \rangle = \langle\psi| H |\psi\rangle \,. \tag{4.26}$$

Quantum optimization algorithms encode the cost function $F(\mathbf{x})$ as a Hamiltonian H_F, such that the ground state of the Hamiltonian (for a minimization problem) corresponds to the optimal solution of the problem [41, 49]. The goal is to find the ground state of the Hamiltonian, which corresponds to the minimum value of the cost function:

$$E_0 = \min_{|\psi\rangle} \langle\psi| H_F |\psi\rangle \,. \tag{4.27}$$

Therefore, if we can encode $F(\mathbf{x})$ as a Hamiltonian H_F, we can minimize F by finding the ground state of H_F. This key idea behind quantum optimization algorithms is motivated by the quantum adiabatic theorem, described below.

A quantum system can be described by a Hamiltonian $H(t)$ that evolves over time. The adiabatic theorem states that if the Hamiltonian is perturbed slowly enough, a quantum system will remain in its instantaneous ground state throughout the evolution [15]. This means that if we start with a quantum system in the ground state of an initial Hamiltonian $H(0)$ and slowly evolve it to a final Hamiltonian $H(T)$, the system will remain in the ground state of $H(t)$ for all times $t \in [0, T]$, mathematically expressed as:

$$H(s) = (1 - s)\, H(0) + sH(T), \tag{4.28}$$

as shown in Figure 4.5, where $s = t/T$. This holds if T is sufficiently large. Additionally, the system must not cross any energy levels during the evolution, which is ensured by the condition that the energy gap between the ground state and the first excited state remains

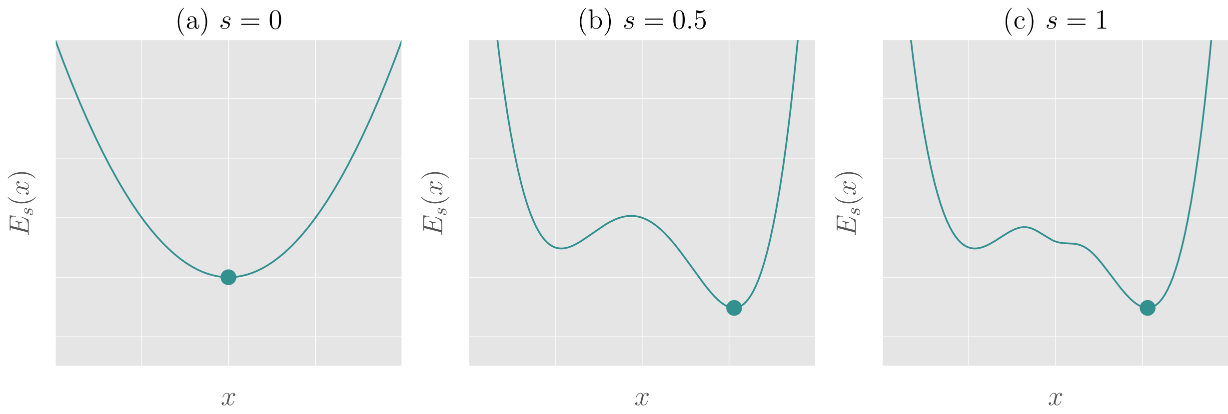


Figure 4.5: Snapshots of the instantaneous energy landscape $E_s(x) = \langle x|H(s)|x\rangle = (1-s)E_0(x) + s\,E_C(x)$ as a function of the computational basis label x. (a) $s = 0$ gives the simple, known Hamiltonian H_0. (b) At $s = 0.5$ small perturbations from the cost Hamiltonian H_C begin to appear. (c) $s = 1$ recovers the full cost Hamiltonian H_C, with its multiple minima.

non-zero throughout the evolution [15].

4.2.1 Encoding Optimization Problems

As described in the previous section, quantum optimization algorithms encode optimization problems as Hamiltonians. The encoded Hamiltonian H_F is constructed such that

$$H_F\,|\mathbf{x}\rangle = F(\mathbf{x})\,|\mathbf{x}\rangle \tag{4.29}$$

for all $\mathbf{x}$, where $F(\mathbf{x})$ is the cost function to be minimized, and $|\mathbf{x}\rangle$ is the quantum state representing the solution $\mathbf{x}$. Therefore, if we are able to encode the cost function $F(\mathbf{x})$ as $H_F(\mathbf{x})$, the optimal solution can be found as:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} F(\mathbf{x}) = \arg\min_{|\mathbf{x}\rangle} \langle\mathbf{x}|\,H_F\,|\mathbf{x}\rangle\,. \tag{4.30}$$

Mapping the cost function to a Hamiltonian is done by the following steps:

1. The variables of the optimization problem are encoded into spin variables by means of a substitution:
$$x_i = \frac{1-z_i}{2}, \quad z_i \in \{-1, 1\}, \tag{4.31}$$
where x_i is the i-th variable of the optimization problem.

2. Starting from the QUBO formulation in Equation (2.3),
$$F(\mathbf{x}) = \sum_{i,j} Q_{ij}x_ix_j,$$

we separate the diagonal and off-diagonal terms:

$$F(\mathbf{x}) = \sum_{i\neq j,\, i,j=1}^{N} Q_{ij} x_i x_j + \sum_{i=1}^{N} \mu_i x_i,$$

where μ_i are the diagonal elements of the QUBO matrix Q, and N is the number of variables. Using Equation (4.31), this QUBO formulation can be expressed in terms of the spin variables z_i as:

$$F(\mathbf{z}) = \sum_{i\neq j,\, i,j=1}^{N} Q_{ij} \left(\frac{1-z_i}{2}\right)\left(\frac{1-z_j}{2}\right) + \sum_{i=1}^{N} \mu_i \left(\frac{1-z_i}{2}\right). \tag{4.32}$$

3. This function is further reformulated as a quantum operator H_F (the cost/problem Hamiltonian) by promoting the spin variables to the Pauli operators Z_i and the constants "1" to the identity operator $\mathbb{1}$:

$$H_F = \sum_{i\neq j,\, i,j=1}^{N} Q_{ij} \left(\frac{\mathbb{1}-Z_i}{2}\right)\left(\frac{\mathbb{1}-Z_j}{2}\right) + \sum_{i=1}^{N} \mu_i \left(\frac{\mathbb{1}-Z_i}{2}\right), \tag{4.33}$$

which on expanding, becomes:

$$H_F = \sum_{i,j,\, i\neq j} \boxed{\frac{Q_{ij}}{4}}\, Z_i Z_j - \sum_i \boxed{\frac{1}{2}\Big(\mu_i + \sum_{j\neq i} Q_{ij}\Big)}\, Z_i + \boxed{\frac{1}{4}\sum_{i,j,\, i\neq j} Q_{ij} + \frac{1}{2}\sum_i \mu_i}\,.$$

The highlighted boxes in the above equation represent the coefficients of the quadratic and linear terms, and the constant term, respectively. This Hamiltonian H_F can be further simplified (as in [50]) by separating out the constant terms, which do not affect the optimization problem, leading to the final form of the Hamiltonian:

$$H_F = \sum_{i,j} W_{ij} Z_i Z_j - \sum_i w_i Z_i + c, \tag{4.34}$$

where we defined $W_{ij} = {Q_{ij}}/{4}$, $w_i = \frac{1}{2}(\mu_i + \sum_j Q_{ij})$, and $c = \frac{1}{4}\sum_{i,j} Q_{ij} + \frac{1}{2}\sum_i \mu_i$. The Hamiltonian H_F written in this form is referred to as the Ising Hamiltonian, due to its similarity to the Ising model in statistical mechanics [15, 35]. The product $Z_i Z_j$ represents the interaction between the qubits i and j, similar to the interaction between spins in the Ising model.

Following the above steps, we can encode any QUBO problem as a Hamiltonian H_F that can be used in quantum optimization algorithms. The ground state of this Hamiltonian corresponds to the optimal solution of the optimization problem, and the energy of the ground state gives the minimum value of the cost function (i.e., the cost value). The Ising

Hamiltonian formulation of many famous combinatorial optimization problems, such as the Maximum Cut problem is well known, and can be seen in References [15, 51].

QAOA is inspired by the adiabatic annealing process, where the Hamiltonian is tuned over time as

$$H(t) = -A(t)\sum_i X_i + B(t)\sum_{i,j} Z_i Z_j, \tag{4.35}$$

where $A(t)$ and $B(t)$ are time-dependent functions that control the evolution of the Hamiltonian. The initial Hamiltonian is typically chosen to be a simple Hamiltonian with a known ground state, while the final Hamiltonian encodes the optimization problem. As seen in Equation (4.35), a typical annealing schedule starts with a Hamiltonian that is dominated by the transverse field term $-A(t)\sum_i X_i$, and then gradually transitions to the cost Hamiltonian $B(t)\sum_{i,j} Z_i Z_j$, which encodes the optimization problem.

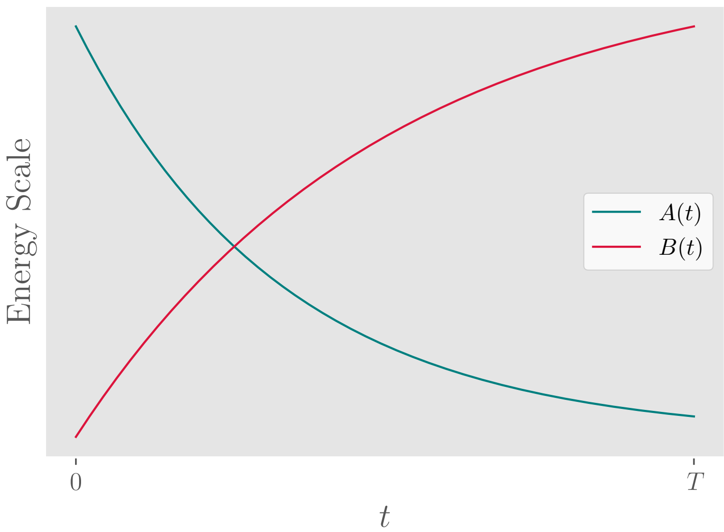


Figure 4.6: A typical adiabatic annealing schedule. The figure shows the evolution of the Hamiltonian from an initial Hamiltonian dominated by the transverse field term scaled by $A(t)$ to a final cost Hamiltonian scaled by $B(t)$.

4.2.2 Variational Quantum Algorithms

Variational quantum algorithms (VQAs) are a class of quantum algorithms that use a parameterized quantum circuit to approximate the solution to a problem [41, 52]. The key idea behind VQAs is to use a quantum circuit with adjustable parameters, which can be optimized to minimize the expectation value of a cost Hamiltonian.

Given the Hamiltonian H encoding an optimization problem, the goal of a VQA is to find the parameters θ that minimize the expectation value of the Hamiltonian:

$$\theta^* = \arg\min_\theta \langle\psi(\theta)| H |\psi(\theta)\rangle, \tag{4.36}$$

where $|\psi(\theta)\rangle$ is an ansatz or trial state prepared by a parameterized quantum circuit with parameters θ. The expectation value $\langle\psi(\theta)| H |\psi(\theta)\rangle$ gives the energy corresponding to the Hamiltonian H and is computed by measuring the quantum state after applying the parameterized circuit, and it represents the cost function to be minimized. This is based on the variational principle in quantum mechanics [53], which states that the expectation value of a Hamiltonian is an upper bound on the ground state energy of the system. In other words, the energy obtained using the parameters in Equation (4.36) as $E(\theta^*)$ is an upper bound on the ground state energy of the Hamiltonian H, and $\psi(\theta^*)$ approximates the ground state of H.

The optimization process Equation (4.36) is itself NP-hard in general [51], and hence it is typically performed in a hybrid quantum-classical manner, where a classical optimizer is used to adjust the parameters θ, while the quantum computer is used to calculate information about the expectation value of the Hamiltonian. The optimization process is repeated until convergence, leading to a set of parameters $\Theta^* = \{\theta_1^*, \theta_2^*, \ldots, \theta_n^*\}$ that minimize the expectation value of the Hamiltonian. The final quantum state $|\psi(\Theta^*)\rangle$ represents an approximate solution to the optimization problem encoded in the Hamiltonian H. Solvers based on VQAs are often referred to as variational quantum eigensolvers (VQEs) [54].

The main advantage of VQAs is that they can be run on near-term quantum devices, which have limited coherence times and are susceptible to noise [52]. This gives VQAs an advantage over a direct implementation of adiabatic quantum computation, which requires a long evolution time to ensure that the system remains in its ground state throughout the evolution.

4.2.3 The Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) is a variational quantum algorithm designed to solve combinatorial optimization problems [13, 14]. QAOA is based on the idea of encoding the optimization problem as a Hamiltonian, similar to the adiabatic quantum computation approach, but it uses a parameterized quantum circuit to approximate the solution. The QAOA generates a quantum state that approximates the ground state of the Hamiltonian H_F encoding the optimization problem, by alternating between applying H_F and a mixing Hamiltonian H_M used to explore the solution space. The algorithm starts with an initial state, typically a uniform superposition of all possible states (or ground state of H_M), and then applies a sequence of gates that encode the cost and mixing Hamiltonians. The standard QAOA algorithm [13, 50, 55] is described below:

1. **Initialization:** The QAOA circuit starts with an initial state, typically a uniform superposition of all possible states, achieved by applying Hadamard gates to all qubits:

$$|\psi_0\rangle = H^{\otimes n} |0\rangle^{\otimes n} = |+\rangle^{\otimes n} \tag{4.37}$$

where $H^{\otimes n}$ denotes the Hadamard gate applied to each of the n qubits, and $|+\rangle$ is the uniform superposition state defined in Equation (4.15). The choice of the initial state is made to be the ground state of the mixing Hamiltonian, which is typically chosen to be the sum of Pauli-X operators (called the standard mixer) [50].

2. **Cost Hamiltonian Application:** The cost Hamiltonian H_F is applied to the quantum state using the time evolution operator induced by H_F during time γ,

$$U_F(\gamma) = e^{-i\gamma H_F},$$

which encodes the problem to be solved. The state after this step is given by:

$$|\psi_1\rangle = U_F(\gamma)\,|\psi_0\rangle = e^{-i\gamma H_F}\,|\psi_0\rangle\,. \tag{4.38}$$

The exponentiation of the cost Hamiltonian can be done as follows: starting from Equation (4.34), we can write the cost Hamiltonian as:

$$H_F = H_{\text{quad.}} + H_{\text{lin.}} + c, \tag{4.39}$$

where $H_{\text{quad.}} = \sum_{i,j,\, i\neq j} W_{ij} Z_i Z_j$ is the quadratic part, $H_{\text{lin.}} = -\sum_i w_i Z_i$ is the linear part, and c is a constant term that does not affect the optimization problem. Omitting this constant, the unitary operator $U_F(\gamma)$ can then be expressed as:

$$U_F(\gamma) = e^{-i\gamma H_F} = e^{-i\gamma H_{\text{quad.}}} e^{-i\gamma H_{\text{lin.}}}.$$

Note that this decomposition is valid only if the two parts of the Hamiltonian commute, i.e., $[H_{\text{quad.}}, H_{\text{lin.}}] = 0$ [28]. The terms do in fact commute in the case of an Ising Hamiltonian, which can be seen from the fact that $[Z_i Z_j, Z_k] = 0$ for all i, j, k, and hence the cost Hamiltonian can be expressed as a product of exponentials of the individual terms. Now, the unitary operator $U_F(\gamma)$ can be expressed as:

$$U_F(\gamma) = \prod_{i,j} e^{-i\gamma W_{ij} Z_i Z_j} \prod_i e^{-i\gamma w_i Z_i}. \tag{4.40}$$

The first term is the R_{zz} gate defined in Equation (4.23), which can be applied to the qubits i and j with parameter $\theta = 2\gamma W_{ij}$, and the second term is a single-qubit rotation gate $R_z(\theta) = e^{-i\frac{\theta}{2}Z}$ applied to the qubit i with parameter $\theta = 2\gamma w_i$. This is illustrated in Figure 4.7, for a three-qubit problem as an example, where the cost layer $U_F(\gamma) = e^{-i\gamma H_F}$ consists of R_{zz} gates applied to all the pairs of qubits (a fully-connected graph), and single-qubit rotation gates applied to each qubit.

3. **Mixing Hamiltonian Application:** The mixing Hamiltonian H_M is now applied to

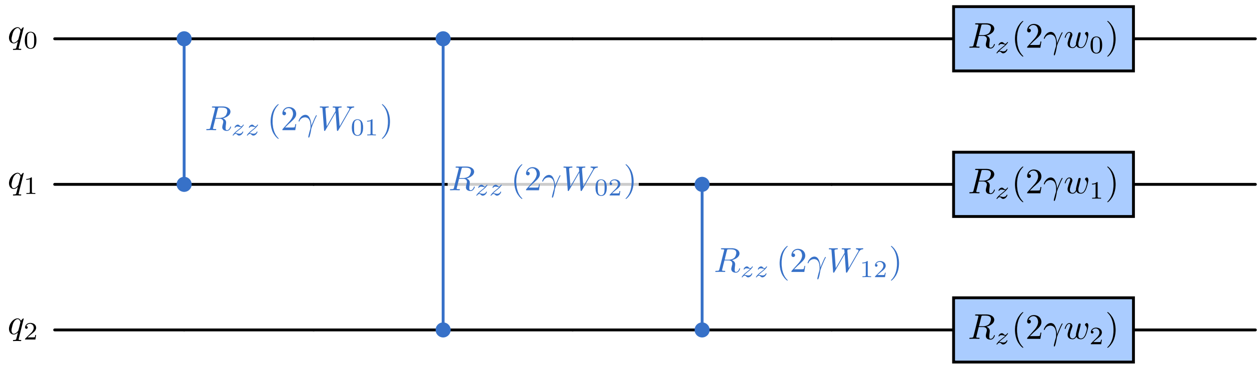


Figure 4.7: Example of the cost layer $U_F(\gamma)$ of the QAOA circuit, for a three-qubit problem. The layer consists of R_{zz} gates applied to pairs of qubits with non-zero coefficients W_{ij} for that pair, corresponding to the quadratic terms of the cost Hamiltonian H_F (see Equation (4.34)), and single-qubit rotation gates applied to each qubit, corresponding to the linear terms w_i of the cost Hamiltonian.

the quantum state, using the time evolution operator induced by H_M during time β,

$$U_M(\beta) = e^{-i\beta H_M},$$

which helps explore the solution space. The state after this step is given by:

$$|\psi_2\rangle = U_M(\beta)\,|\psi_1\rangle = e^{-i\beta H_M} U_F(\gamma)\,|\psi_0\rangle = e^{-i\beta H_M} e^{-i\gamma H_F}\,|\psi_0\rangle\,. \tag{4.41}$$

The mixing Hamiltonian in the standard version of QAOA is typically chosen to be the Pauli-X operator:

$$H_M = -\sum_{i=1}^{N} X_i \tag{4.42}$$

where X_i is the Pauli-X operator acting on the i-th qubit. This Hamiltonian generates the unitary operator $U_M(\beta)$ as:

$$U_M(\beta) = e^{-i\beta H_M} = \prod_{i=1}^{N} e^{-i\beta X_i} = \prod_{i=1}^{N} R_x(2\beta), \tag{4.43}$$

where $R_x(\theta) = e^{-i\frac{\theta}{2}X}$ is the single-qubit rotation gate around the x-axis of the Bloch sphere, with parameter $\theta = 2\beta$. This is illustrated in Figure 4.8, for a three-qubit problem, where the mixing layer consists of single-qubit rotation gates applied to each qubit.

This step completes one iteration of the QAOA circuit (Figure 4.9), where the quantum state is transformed by applying the cost layer followed by the mixing layer.

4. **Iteration:** To make the approximation more accurate, steps 2 and 3 are repeated p times, where p is called the iteration depth of the QAOA circuit. In principle, one can

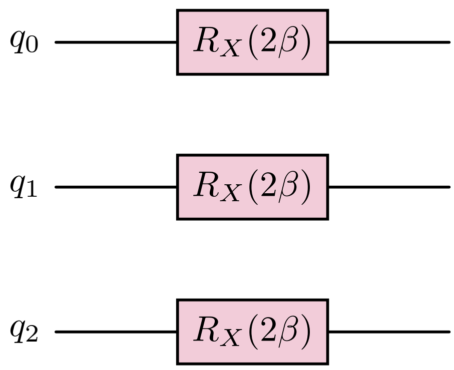


Figure 4.8: Example of the mixing layer $U_M(\beta)$ of the QAOA circuit, for a three-qubit problem. The layer consists of single-qubit rotation gates applied to each qubit, corresponding to the mixing Hamiltonian. The rotation gates are applied with parameter $\theta = 2\beta$, where β is the parameter associated to the mixing Hamiltonian.

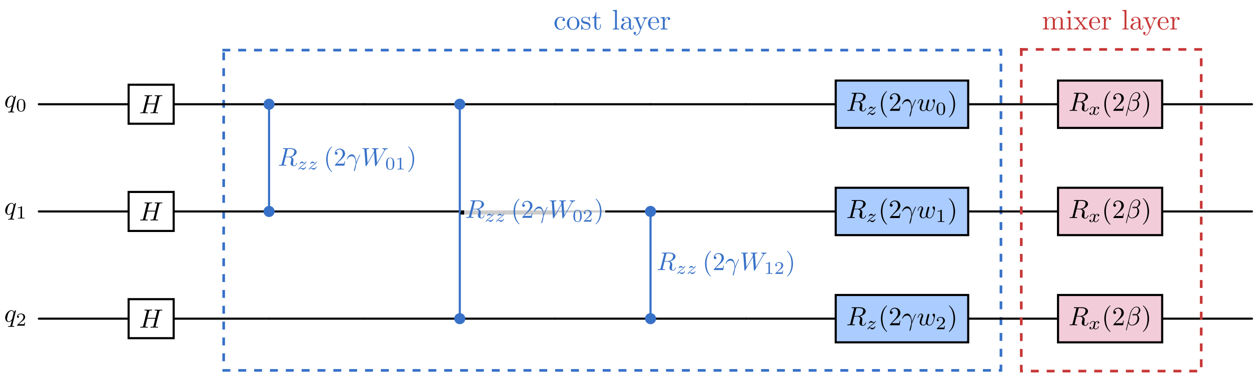


Figure 4.9: Example of a single layer of the QAOA circuit for a three-qubit fully connected problem. Hadamard gates are applied to all qubits to create a uniform superposition state, followed by the cost Hamiltonian layer (consisting of R_{zz} gates and single-qubit R_z gates) and the mixing Hamiltonian layer (consisting of single-qubit R_x gates). The circuit is parameterized by γ and β, which are the parameters associated to the cost and mixing Hamiltonians, respectively.

attain arbitrarily high accuracy by increasing the number of repetitions p [13, 56], but in practice, this is limited by the noise and errors in the quantum device. The final state after p repetitions is given by:

$$\left|\psi_{\vec{\gamma},\vec{\beta}}\right\rangle = \underbrace{U_M(\beta_p)U_F(\gamma_p)\cdots U_M(\beta_1)U_F(\gamma_1)}_{p \text{ iterations}} |\psi_0\rangle\,, \tag{4.44}$$

where $\vec{\gamma} = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ and $\vec{\beta} = \{\beta_1, \beta_2, \ldots, \beta_p\}$ are the parameters of the QAOA circuit.

5. **Measurement:** The final quantum state $|\psi_p\rangle$ is measured in the computational basis, yielding a bitstring that represents a candidate solution to the optimization problem.

More specifically, the measurement gives a classical bitstring $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where $x_i \in \{0, 1\}$ represents the value of the i-th variable in the optimization problem. The probability of measuring a particular bitstring $\mathbf{x}$ is given by:

$$P(\mathbf{x}) = |\langle \mathbf{x} | \, \psi_p \rangle|^2. \tag{4.45}$$

The measured bitstring $\mathbf{x}$ is then used to evaluate the cost function $F(\mathbf{x})$, and the process is repeated to sample multiple solutions. The expectation value of the cost Hamiltonian H_F is then computed as:

$$\langle H_F \rangle = \sum_{\mathbf{x}} P(\mathbf{x}) F(\mathbf{x}) = \sum_{\mathbf{x}} |\langle \mathbf{x} | \, \psi_p \rangle|^2 F(\mathbf{x}). \tag{4.46}$$

This expectation value is passed to a classical optimizer to get better parameters for the next run of the QAOA circuit.

6. **Parameter Optimization:** The parameters $\vec{\gamma}$ and $\vec{\beta}$ are optimized to minimize the expectation value of the cost Hamiltonian H_F:

$$\Theta^* = \arg\min_{\Theta} \langle \psi_p | \, H_F \, | \psi_p \rangle \,, \tag{4.47}$$

This means that the next run of the QAOA circuit will use the updated parameters $\vec{\gamma}^* = \{\gamma_1^*, \gamma_2^*, \ldots, \gamma_p^*\}$ and $\vec{\beta}^* = \{\beta_1^*, \beta_2^*, \ldots, \beta_p^*\}$, which are obtained from the classical optimization process. Hence, this step is basically a variational optimization step, where the parameters of the quantum circuit are adjusted to minimize the expectation value of the cost Hamiltonian.

This entire process is repeated until convergence, leading to a set of parameters $\Theta^* = \{\gamma_1^*, \gamma_2^*, \ldots, \gamma_p^*, \beta_1^*, \beta_2^*, \ldots, \beta_p^*\}$ that minimize the expectation value of the cost Hamiltonian H_F. The final quantum state $|\psi_p(\Theta^*)\rangle$ represents an approximate solution to the optimization problem encoded in the Hamiltonian H_F. This hybrid quantum-classical optimization process is summarized in Algorithm 4.1 and illustrated in Figure 4.10, where the circuit consists of p layers of alternating cost and mixing Hamiltonians.

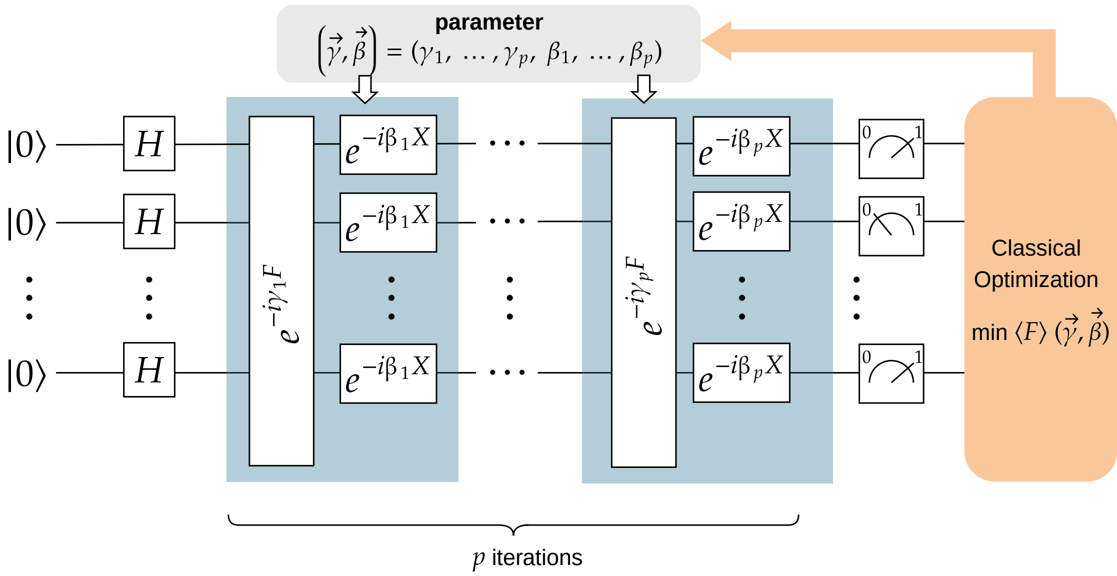


Figure 4.10: Quantum Approximate Optimization Algorithm (QAOA) circuit. The circuit initially consists of Hadamard gates applied to all qubits, creating a superposition of all possible states. The circuit then alternates between applying the cost unitary $U_F(\gamma) = e^{-\gamma F}$ and the mixing unitary $U_M(\beta) = e^{-i\beta X}$. The cost Hamiltonian encodes the problem to be solved, while the mixing Hamiltonian helps explore the solution space. The parameters γ and β are optimized to minimize the expectation value of the cost Hamiltonian, leading to a solution that approximates the optimal solution of the problem. The final measurement collapses the quantum state into a classical bitstring representing a candidate solution.

Algorithm 4.1: Hybrid QAOA Optimization (depth-p)

Input: p (number of QAOA layers), N_{shots} (shots per evaluation),
initial angles $\boldsymbol{\theta}^{(0)} = (\gamma_1^{(0)}, \ldots, \beta_p^{(0)})$,
optimizer settings (max iters, tolerances)

Output: Optimized angles $\boldsymbol{\theta}^*$

$k \leftarrow 0$;

repeat

  $\mathcal{C} \leftarrow \text{BuildCircuit}(\boldsymbol{\theta}^{(k)}, p)$;

  $f_k \leftarrow \text{MeasureCost}(\mathcal{C}, N_{\text{shots}})$;

  $\boldsymbol{\theta}^{(k+1)} \leftarrow \text{UpdateAngles}(\{(\boldsymbol{\theta}^{(i)}, f_i)\}_{i \leq k})$;

  $k \leftarrow k + 1$;

until *convergence or max iterations reached*;

return $\boldsymbol{\theta}^{(k)}$ as $\boldsymbol{\theta}^*$

4.2.4 Fixed parameter schedules: Linear Ramp QAOA

As seen in Section 4.2.3, the QAOA algorithm involves optimizing the parameters γ and β to minimize the expectation value of the cost Hamiltonian. However, this optimization can be challenging, especially for larger problem instances. Firstly, the optimization landscape of the parameters can be computationally expensive to evaluate. A QAOA circuit with p layers requires p number of β and p number of γ parameters, leading to a total of $2p$ parameters to optimize. This is a hard variational problem [51], especially for larger problem instances. Secondly, although a choice of a bigger p can in principle lead to better approximations of the optimal solution [13, 56], this inevitably increases the depth of the circuit, which can lead to increased noise and errors in the quantum device. This is particularly relevant for NISQ devices, which have limited coherence times and are susceptible to noise (as mentioned in Section 4.2.2). The Linear-Ramp QAOA (LR-QAOA) [19, 20] is a method to address the

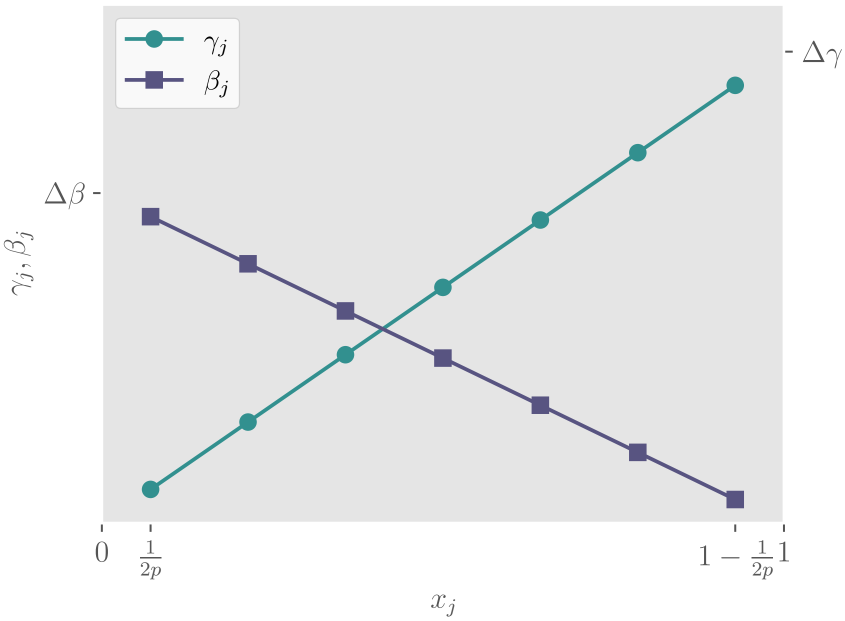


Figure 4.11: Linear ramp ansatz for the QAOA parameters γ and β. The figure shows the linear ramp for a QAOA circuit with $p = 7$ layers, where the parameters γ_j and β_j are linearly spaced based on the values of x_j. The parameters are set to fixed values according on the linear ansatz, which simplifies the optimization process.

first challenge by the use of a fixed parameter schedules, where the parameters γ and β are not optimized during the execution of the algorithm, but rather are set to fixed values chosen based on a linear ansatz (Figure 4.11):

$$\begin{aligned} \gamma_j &= \Delta_\gamma x_j, \\ \beta_j &= \Delta_\beta \left(1 - x_j\right), \end{aligned} \tag{4.48}$$

with $x_j \in [0, 1]$ being a set of p parameters that are uniformly spaced:

$$x_j = \frac{j - {}^1\!/\!_2}{p} \quad \text{for } j = 1, 2, \ldots, p. \tag{4.49}$$

This approach simplifies the optimization process and reduces the number of parameters to optimize, from $2p$ (in the standard QAOA) to just two parameters, Δ_γ and Δ_β, which can be seen as quantities representing slopes of the linear ramp for γ and β, respectively. This linear ansatz is motivated by the adiabatic schedule in Figure 4.6. The parameters γ and β are set to fixed values based on the linear ramp so that, just as in the adiabatic annealing process, the cost Hamiltonian is slowly ramped up while the 'known' Hamiltonian (here the mixing Hamiltonian) is slowly ramped down.

The details of the LR-QAOA method and determination of the Δ_γ and Δ_β parameters is discussed in Chapter 5.

Chapter 5

Scaling of Classical and Quantum Algorithms

Having established the theoretical foundations of combinatorial optimization along with the corresponding classical and quantum algorithms, this chapter presents the methods and experimental setup used to evaluate their performance on the problems of interest: MaxCut and Technician-Asset Allocation (TA). The focus is on comparing the different approaches on two levels: first, a comparison among classical (and quantum) algorithms with each other to understand their scaling and performance characteristics; and second, a comparative analysis of quantum algorithms against their classical counterparts.

The chapter begins with a description of the experimental setup, outlining the software and hardware employed for analysis. Section 5.2 then describes the instance and parameters chosen for running an asymptotic analysis of the runtime of classical algorithms. Further, Section 5.3 describes the quantum algorithms used to optimize the same problems but for a narrower range of problem sizes. The chapter concludes with a comparison of the scaling of the classical and quantum algorithms in Section 5.5.

5.1 Setup of algorithms

When it comes to evaluating the performance of classical algorithms outlined in Chapter 3, one can in principle study the scaling for large problem sizes (up to hundreds of bits, except for the brute-force algorithm). In contrast to this, since the quantum algorithms are evaluated on a simulator, the problem size is limited (due to memory restrictions) by the number of qubits that can be simulated (an evaluation of these algorithms on NISQ devices would also be limited by the number of qubits). Therefore, when comparing the scaling between different classical algorithms, we consider problem sizes of over $N = 100$, while for the quantum algorithms, the problem sizes are limited to $N \leq 28$ qubits.

All the algorithms were implemented in Python, using standard libraries such as `NumPy` [57], and `SciPy` [58]. For creating and managing graph objects, the package `NetworkX` [59] was used, and `CVXPY` [60, 61] was utilized to implement the Goemans-Williamson algorithm. The CPLEX optimizer [8] was accessed through the Decision Optimization CPLEX Python API [42], which allows for solving linear and mixed-integer programming problems efficiently. Similarly, the heuristics provided by `MQLib` [9] were also implemented using its official Python implementation. All quantum algorithms were implemented using the `Qiskit` [24] framework developed by IBM, which provides a comprehensive set of tools for quantum computing, including circuit construction, simulation, and execution on quantum hardware.

CPLEX and LR-QAOA experiments were executed on an HPC cluster. In contrast, `MQLib` heuristics were run on a local machine (`Win 10 Pro 22H2`; `i7-1365U 10C/12T 1.8 GHz`; `Iris Xe 2 GB`; `16 GB RAM`) because the Python implementation of `MQLib` (supports only Python ≤ 3.8) was incompatible with the cluster software stack. Consequently, `MQLib` wall times are not directly comparable to those obtained on the cluster hardware. This discrepancy in hardware environment does not affect the validitiy of the comparative study (as long as all the instances are run using the same hardware setup), because our interest lies in how each algorithm scales with problem size, rather than the absolute performance on a specific hardware setup. In other words, the comparative analysis does not try to answer the question of which algorithm is faster in absolute terms, but rather how the performance of each algorithm changes as the problem size increases.

To solve the QUBO problems as described in Chapter 3, we start by generating the problem instances for MaxCut and TA problems. While the MaxCut QUBOs are generated from the adjacency matrices of the corresponding graphs (Equation (2.7)), the TA QUBOs are generated from the distance matrices obtained from the use-case generator provided by the project partner EnBW (Equation (2.16) and Section 5.2.2). The details of the instance generation are described in Section 5.2. In all cases, we set up the QUBO matrix so that the problem is a minimization problem, i.e., the objective function is to be minimized.

To solve a problem using CPLEX, we first convert the QUBO matrix into a doCPLEX model object [42]. The model object requires an objective function and optimization method (maximize/minimize) to be specified. The objective function is constructed from the QUBO matrix by iterating over the upper triangular part of the matrix (since it is symmetric) and adding the corresponding terms to the objective function. The optimization method is set to 'minimize' and the model is then solved using the `solve()` method.

Qiskit provides a limited set of classical optimizer programs that can be used to solve QUBO problems [24]. We use the `GoemansWilliamsonOptimizer` class provided in this package, which implements the GW algorithm for MaxCut problems. Similar to CPLEX's objective function construction, the QUBO matrix is converted into a quadratic program (QP) object to be used in Qiskit's GW implementation. In addition to this, the Qiskit optimizer further takes in an integer '`num_cuts`' which specifies the number of cuts to be considered in the randomized rounding step of the GW algorithm (see Section 3.3.2). A higher value of '`num_cuts`' leads to a more accurate approximation of the MaxCut solution, but also increases the runtime of the algorithm. In all the experiments, we set this parameter to a fixed value, i.e., '`num_cuts`' = 5.

The `MQLib` solver takes in a QUBO matrix, the choice of heuristic, and a time bound parameter to limit the runtime of the solver. The heuristic is specified as a string, and the time bound is set to a fraction of the problem size ($\alpha \cdot N$, $\alpha \in [0, 1]$ seconds). The solver then returns the history of the best cost function values found during the optimization process and the corresponding time taken to reach each solution.

5.2 Classical Optimization Results

This section presents the results of the classical optimization algorithms (see Chapter 3) applied to the problems of interest: MaxCut and Technician-Asset Allocation (TA). The results are organized by problem type, with a focus on the scaling behaviour of the algorithms and their performance on different problem instances.

5.2.1 Performance on MaxCut Instances

The MaxCut problem is evaluated using various classical algorithms, including Goemans-Williamson, CPLEX, and `MQLib` heuristics. Before presenting the results, we briefly describe the problem instances set up for evaluating the algorithms.

To study the optimization of the MaxCut problem, we consider weighted graphs with varying numbers of nodes N. The exclusion of unweighted graphs was motivated by the fact that the MaxCut problem for an unweighted graph can have multiple solutions with the same cut value, which means that there might not be just one optimal solution. For example, for an unweighted complete graph with N nodes, the number of optimal solutions increases polynomially with N (any bitstring with Hamming weight $N/2$ is an optimal solution). This makes it difficult to compare the performance of different algorithms, as they might find different optimal solutions. Therefore, we focus only on weighted graphs, where we assign weights to the edges from a uniform distribution. Note that, since the MaxCut problem is symmetric, there will still be two optimal solutions for a given graph, as seen in Figure 2.9.

In order to analyse the asymptotic behaviour of the runtime, we look at the scaling of the algorithms for weighted complete graphs and weighted random-3-regular graphs (see Section 2.3.2; here 'random' means that the edges for the 3-regular graph is sampled from a uniform distribution), with weights drawn from a uniform distribution between $[1, 1000]$. The instances are generated for $N = 10, 20, \ldots, 200$ nodes, and for each N, we generate 10 random instances to ensure a representative sample of varying graph structures.

Although CPLEX offers excellent performance for unweighted problems and small-scale weighted problems, it was found to scale poorly for large weighted instances. As seen in Section 3.2, CPLEX uses a 'gap' parameter to determine when to stop the optimization process, which can lead to varying runtimes for different instances. This parameter measures how close the best integer solution found ('incumbent') is to the best bound guaranteed by

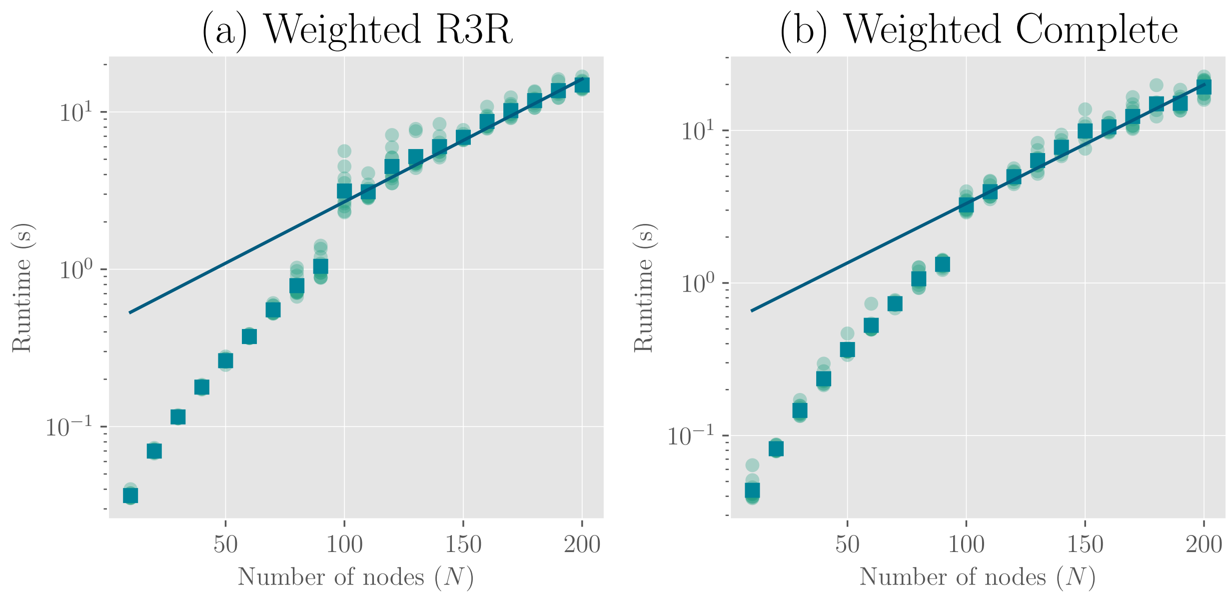


Figure 5.1: Asymptotic scaling of GW runtime for MaxCut problem for (a) weighted random-3-regular (R3R) graphs, and (b) weighted complete graphs. The dots represent the runtimes for each of the 10 instances generated at each N, the square markers represent the geometric mean of the runtimes for each N, and the solid line is the exponential fit to the geometric mean. The best-fit parameters to the function $f(N) = a \cdot 2^{bN}$ are $a = 0.445$, $b = 0.026$ for the weighted R3R graphs, and $a = 0.553$, $b = 0.026$ for the weighted complete graphs. Although the Goemans-Williamson algorithm is a polynomial-time approximation scheme, the scaling for large instances fit better to an exponential function.

the solver's relaxation. A gap of 0.01 means that the solver will stop when the incumbent solution is within 1% of the best bound. For weighted MaxCut problems, the CPLEX branch-and-bound explodes in the number of children nodes to be explored, leading to very long runtimes. For a 50-vertex graph instance, the CPLEX solver explored over 2×10^5 nodes, while on a 90-vertex instance, it explored over 10^7 nodes before even reaching a 2% gap (see Appendix B). Hence it was seen that CPLEX did not offer reliable performance for large weighted MaxCut problems, and therefore was excluded from the asymptotic analysis.

The asymptotic scaling of the the Goemans-Williamson algorithm and the `MQLib` heuristics is shown in Figures 5.1 and 5.3, with the geometric mean of the runtimes for each N plotted along with the fit to an exponential function of the form $f(N) = a \cdot 2^{bN}$. The `MQLib` solver was executed with the `BURER2002` heuristic (see Section 3.4), with a timebound for each instance kept as a fraction of the problem size (empirically determined `timebound` $= 0.27 \times N$ seconds). The runtime scaling for the GW algorithm shows a clear step-up in scaling at $N = 100$, which could be attributed to some internal thresholds being crossed in `CVXPY`'s SDP solver, leading to a change in the scaling behaviour. However, the objective values returned by the GW algorithm for MaxCut instances fall in quality for increasing N, in comparison to the `MQLib`, as can be seen in Figure 5.4. Interestingly, the geometric mean data in both regimes (before and after $N = 100$) fit well to a polynomial function of the form $f(N) = a + bN^p$ as well as to an exponential function of the form $f(N) = a \cdot 2^{bN}$ (see Figure 5.2). The degree p in the polynomial fit and the parameter b in the exponential fit are both indicators of the scaling behaviour of the algorithm, with a higher value indicating a steeper increase in runtime with increasing problem size. The best-fit parameters for the polynomial and exponential fits are shown in the figure legend.

The results indicate the major observation that, while both the GW and `MQLib` solvers scale comparable to each other in the problem set studied, the `MQLib` heuristics find solutions with a higher cut value than the GW algorithm for larger problem sizes (Figure 5.4). Note that the quality of solution for the GW algorithm can be improved by increasing the number of cuts generated at the randomized rounding step (which Qiskit's implementation controls using an input parameter). However, in our experiments, even when assigning this parameter to scale linearly with the input size (see Section 5.5), the quality of the solution did not show significant improvement in comparison to the other solvers. This is an important observation, as it suggests that the Burer algorithm (see Section 3.4) implementation by `MQLib` can be more effective than the GW algorithm for large MaxCut instances, despite the latter being a well-known approximation scheme. However, it is important to note that the `MQLib` heuristics are also not guaranteed to find the optimal solution, and therefore the cut values obtained from them still may not be the best possible one. Therefore in the smaller instance set

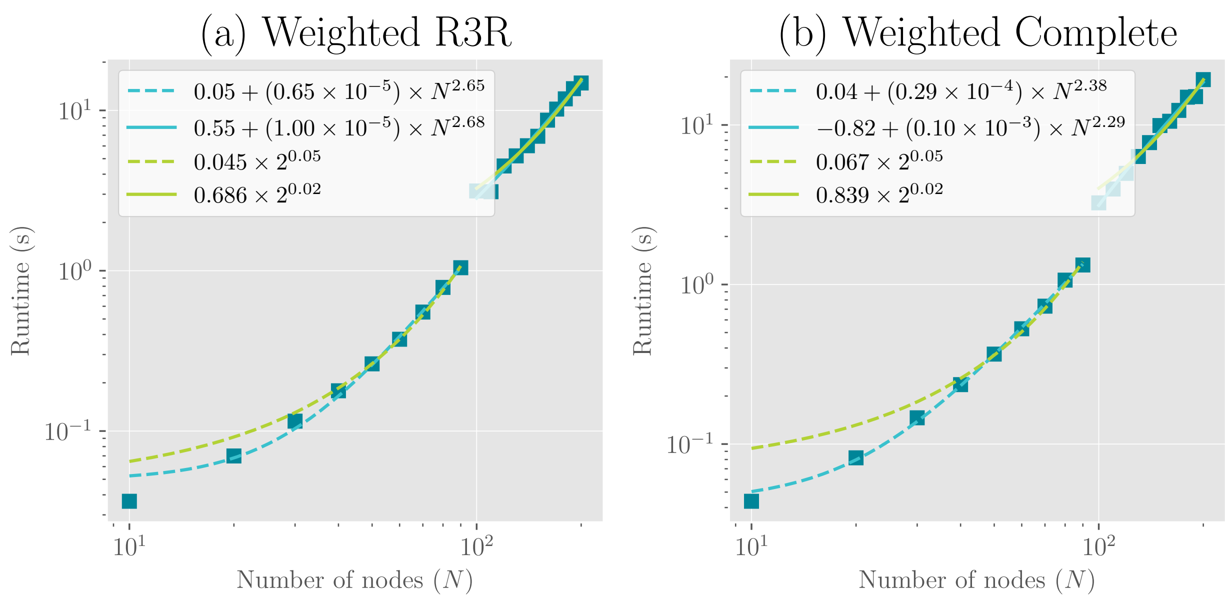


Figure 5.2: Piece-wise fit to the geometric mean runtimes of GW algorithm for MaxCut problem for (a) weighted R3R graphs, and (b) weighted complete graphs. Power law fits (dashed lines for $N \leq 100$ and solid lines for $N > 100$) highlight the change in scaling behaviour at $N = 100$. Both regimes fit well to a power law of the form $f(N) = a + b \cdot N^p$, as well as to an exponential function of the form $f(N) = a \cdot 2^{bN}$, whose best fit parameters are shown in the legend.

up for comparing with quantum algorithms (Section 5.5), we set the runtime bound to be sufficiently large ($\approx 10N$ seconds at each N) to ensure that the `MQLib` heuristics can find the optimal solution, and as a confirmatory step, we verify the solution obtained by CPLEX, setting the gap parameter to a very small value ($\sim 10^{-6}$).

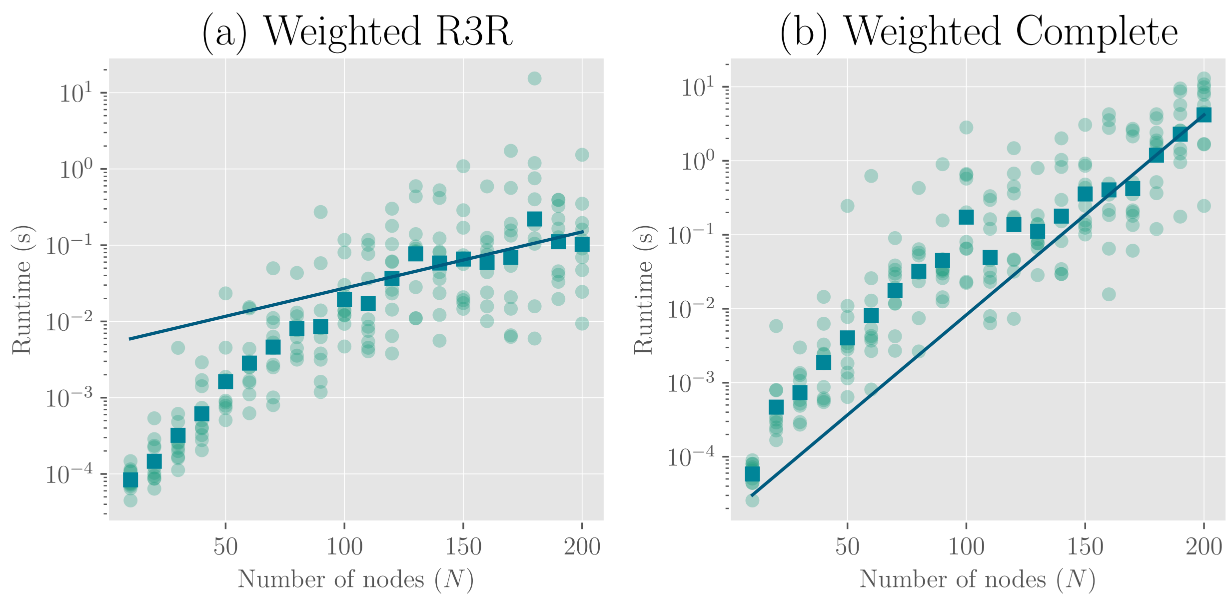


Figure 5.3: Same as Figure 5.1 but for `MQLib`. The best-fit parameters to the function $f(N) = a \cdot 2^{bN}$ are $a = 4.99 \times 10^{-3}$, $b = 0.025$ for the weighted R3R graphs, and $a = 1.63 \times 10^{-5}$, $b = 0.089$ for the weighted complete graphs.

5.2.2 Performance on Technician-Asset Allocation Instances

The instances for the Technician-Asset Allocation (TA) problem were generated using a use-case generator provided by the project partner EnBW. The generator starts by querying the Overpass API [62] for all power-poles and the administrative boundary of the district of Rottweil, Germany. It then assigns a specified T number of those power-poles as technicians, and the remaining poles as assets, avoiding any overlap between technicians and assets, and assigning unique IDs to each technician and asset. The locations of all the queried points are then saved as a `csv` file, which is then used to compute the distances between each technician-asset pair using the Haversine formula [63], thus generating a distance matrix. The longest Technician-Asset separation distance (which we henceforth refer to as $D_{\max}$) in each set is also additionally printed as an output. The original generator also generates an integer to specify the minimum number of technicians each team should have.

The above use-case generator was modified in the scope of this thesis to generate instances with an equal number of technicians and assets, each iterating from values in the range $[2, 3, \ldots, 25]$, so that the resulting QUBOs have problem sizes of $N = T + A = [4, 6, \ldots, 50]$ bits. In other words, at any QUBO instance, the number of technicians and assets are

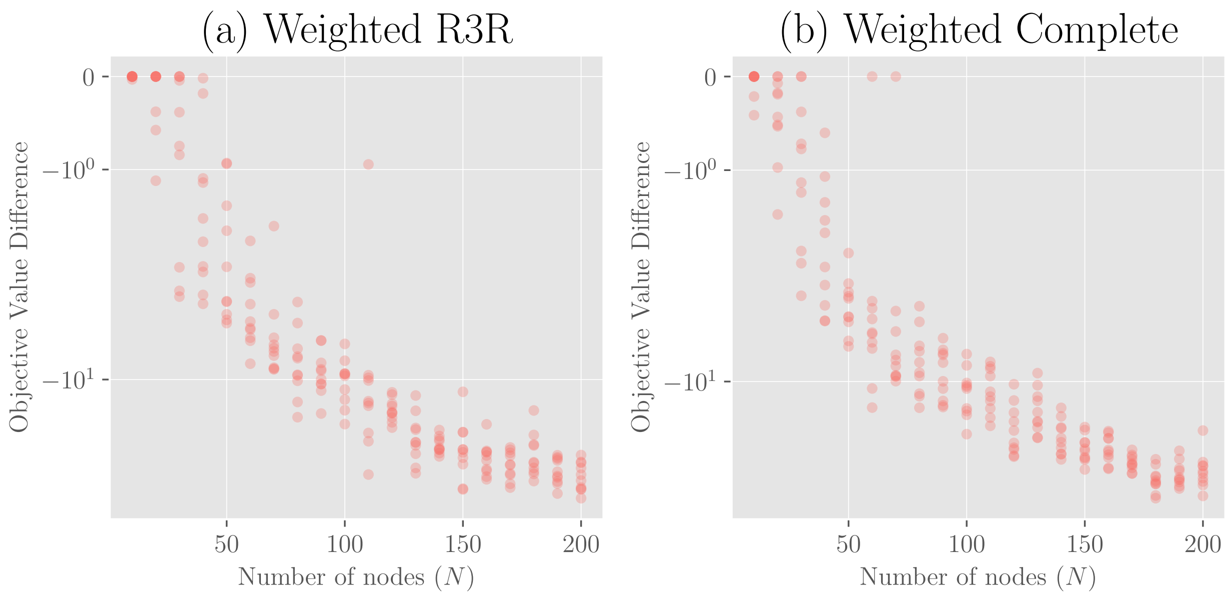
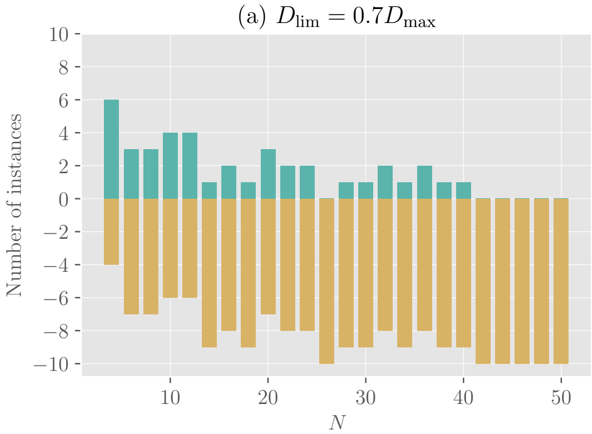


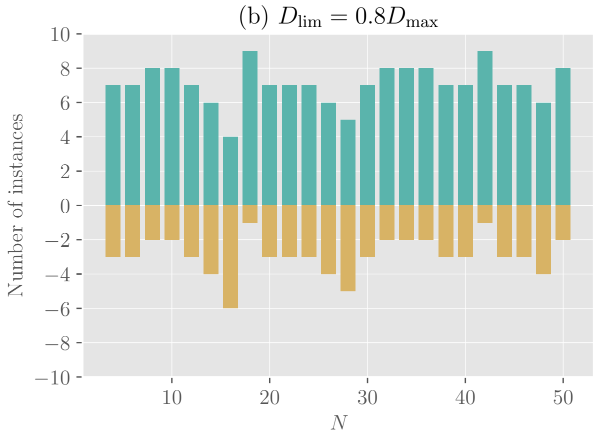
Figure 5.4: Comparison of objective values obtained by `MQLib` heuristics and Goemans-Williamson algorithm for MaxCut problem, plotted as the GW objective value minus the `MQLib` objective value. The increasing negative values with increasing N indicate that the `MQLib` heuristics find solutions with a higher cut value than the GW algorithm for larger problem sizes.

equal, and is equal to $N/2$. Further, the minimum number of technicians T_1 in one team was set to $\lfloor T/2 \rfloor$ and the minimum number of assets A_1 in one team was set to $\lfloor A/2 \rfloor$ (see Section 2.3.3 for definitions of these variables). This was done to enable the generation of sub-samples from a problem instances, a key requirement for the extrapolation method used in the context of the LR-QAOA algorithm (Section 5.3.2).

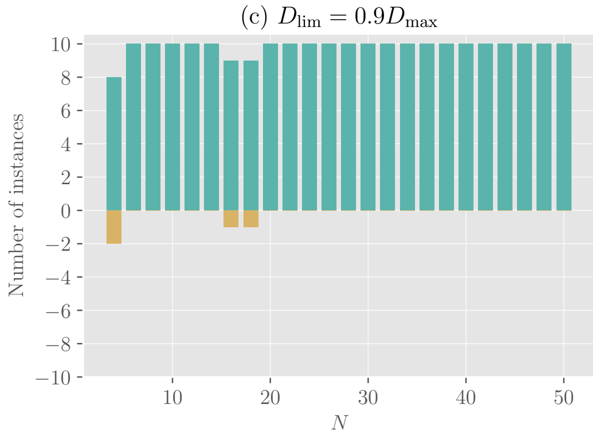
The remaining parameter to be fixed for the QUBO generation is the limiting distance D_{lim} that directly affects the constraints in the problem formulation. This parameter physically represents the maximum distance a technician can travel within a team. The closer this value is to D_{max}, the easier it is to satisfy the feasibility criterion $F \leq 2D_{\text{lim}}$ (see Section 2.3.3). We therefore tested for the number of feasibile versus infeasible solutions obtained for increasing D_{lim} values, setting them as a fraction of D_{max}, specifically $D_{\text{lim}} = f \cdot D_{\text{max}}$, where $f \in \{0.7, 0.8, 0.9\}$. Starting with the first f value, we solved the QUBO for ten random instances at each N using the `MQLib` solver, setting the runtime bound to be $3 \times (0.59 \times N)$ seconds (the recommended runtime bound for the `MQLib` heuristics is $0.59 \times N$ seconds [9]), so that a solution close to the optimal one was obtained. The best solution found by the `MQLib` heuristics is then checked for feasibility for each instance. The results of this check are shown in Figure 5.5. Clearly, the number of feasible solutions increases with increasing D_{lim}, as expected. The results also indicate that the `MQLib` heuristics are able to find feasible solutions for most of the instances at $D_{\text{lim}} = 0.9D_{\text{max}}$, while at lower values of D_{lim}, the number of infeasible solutions increases significantly.



(a)



(b)



(c)

Figure 5.5: Feasibility check for increasing D_{lim} values in the TA problem. Feasible solutions are those which fulfill the constraints, as given in Equation (2.15). Out of the ten random instances evaluated at each N, the green bars (positive counts) indicate the number of instances for which the best solution found by `MQLib` heuristics is feasible, while the yellow bars (negative counts) indicate the number of instances for which the best solution is infeasible. The three subfigures correspond to D_{lim} values (a) $0.7D_{\mathrm{max}}$, (b) $0.8D_{\mathrm{max}}$, and (c) $0.9D_{\mathrm{max}}$. The results indicate that as D_{lim} increases, the number of feasible solutions also increases.

For the analysis of classical scaling, we therefore set $D_{\mathrm{lim}} = 0.9D_{\mathrm{max}}^{(\mathrm{region})}$, which is the longest distance found in the entire dataset. This distance approximately corresponds to the size of the district, which ensures that the best solution at each instance is feasible for most instances, while still maintaining a reasonable distance limit for technician-asset assignments. As a result, it was found that all of the instances satisfy the feasibility criterion with this value of D_{lim}. The scaling of instances from $N = 4, \ldots, 50$ bits were studied, with the number of technicians and assets both ranging from 2 to 25. Since the GW algorithm is not typically used for constrained problems like TA, we only consider the `MQLib` and CPLEX algorithms for the scaling analysis. The `MQLib` solver was executed with the hyper-heuristic

(HH) parameter instead of specifying a particular heuristic as in the MaxCut case. This setting automatically chooses the best heuristic for the given instance. This choice was made keeping in consideration the novelty of the problem, as the HH parameter is designed to adapt to different problem structures [9]. In all cases, MQLib found the same optimal solution as CPLEX.

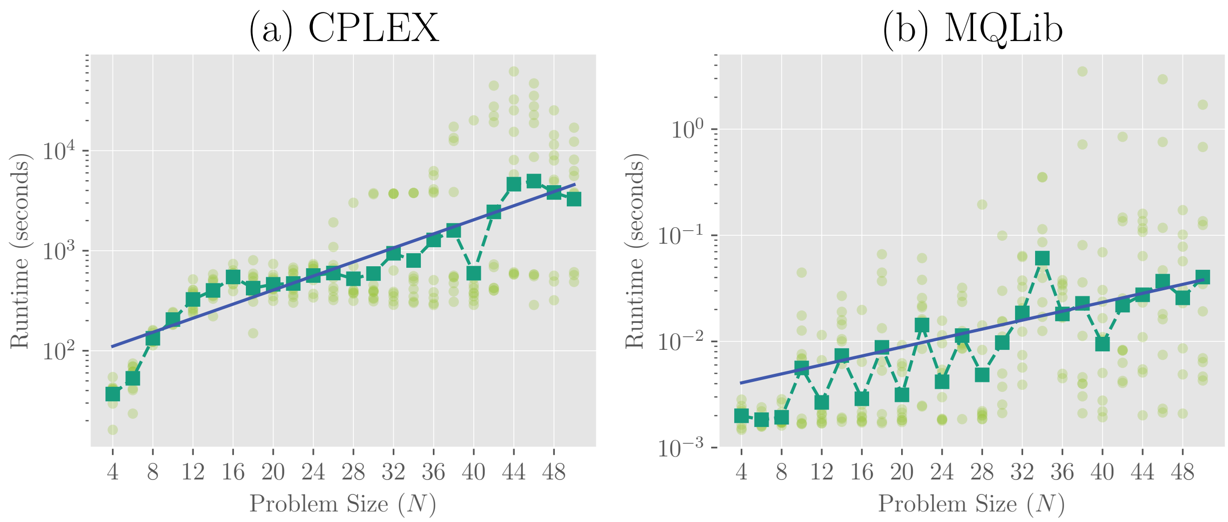


Figure 5.6: Asymptotic scaling of MQLib heuristics and CPLEX for the Technician-Asset Allocation problem, fixing the D_{lim} value to $0.9D_{\text{max}}^{(\text{region})}$. With this D_{lim}, the solutions to all the instances were found to satisfy the constraint. The dots represent the runtimes for each of the 10 instances generated at each N, the square markers represent the geometric mean of the runtimes for each N, and the solid line is the exponential fit to the geometric mean. The best-fit parameters to the function $f(N) = a{\cdot}2^{bN}$ are $a_{\text{CPLEX}} = 80.14$, $b_{\text{CPLEX}} = 0.12$ and $a_{\text{MQLib}} = 3.34 \times 10^{-3}$, $b_{\text{MQLib}} = 0.07$. The results indicate that the MQLib heuristics scale much better than the CPLEX solver for the TA problem, with the former taking significantly longer to solve larger instances.

The scaling of the MQLib heuristics and CPLEX solver is shown in Figure 5.6. In this problem also, MQLib continues to outperform CPLEX, which has significantly higher (absolute) runtime values. This is because MQLib reports 'good-enough' solutions as it obtains them, while CPLEX continues to search for the optimal solution until it reaches the specified gap parameter (which was set to a very small value, $\sim 10^{-6}$). This means that, although CPLEX takes longer to solve larger instances, it is able to find the optimal solution for all instances. It can also be observed that the scaling behaviour of the CPLEX solver is not regular, indicating that the solver adapts to the size of the problem. Hence, the scaling of the runtime is not guaranteed to follow the same behaviour at larger problem sizes.

5.3 Quantum Optimization Procedure

In this section, we describe how we implement the quantum algorithms for the MaxCut and Technician-Asset Allocation (TA) problems, and present the results of their performance. The main algorithm that we focus on this work is the LR-QAOA algorithm (Section 4.2.4), which is a modification of the standard QAOA algorithm. We consider two variants of the LR-QAOA algorithm: the extrapolation-based method introduced in Reference [20], and the normalized Hamiltonian approach introduced in Reference [19]. Starting with a brief description of the two quantum algorithms used, we highlight the difference between the two approaches. We then present the methods used to implement these algorithms, followed by the results of their performance on the MaxCut and TA problems. Unless otherwise specified, the description of the two LR-QAOA variants follows the original papers [20, 19] respectively.

With the same goal as in the classical algorithms, we aim to study the scaling of the runtime of the LR-QAOA algorithm for MaxCut and TA problems. The runtime of both LR-QAOA algorithms studied in this work is quantified by the 'total depth' of the circuit (as in [20]), which is defined as

$$D = d \cdot N_{\text{shots}}, \tag{5.1}$$

where N_{shots} is the median of the number of shots needed to sample the output of the circuit until the optimal solution is found (see also Equation (5.5) below), and d is the depth of the QAOA circuit (which is in turn proportional to the number of layers p). The actual 'time' taken to run the LR-QAOA algorithm is then given by the product of the total depth D and the time taken to execute one gate (single and multi-qubit gates) on the specific quantum hardware, in addition to the time taken to prepare the state and measure the output.

As seen in Section 4.2.4, the LR-QAOA algorithm aims to avoid the variational parameter optimization step of the QAOA algorithm by using a linear ansatz to fix the β and γ parameters (Equation (4.48)), thus reducing the number of parameters to be optimized for a p-layer QAOA circuit from $2p$ parameters $(\gamma_1, \gamma_2, \ldots, \gamma_p \ ; \ \beta_1, \beta_2, \ldots, \beta_p)$ to just two parameters $(\Delta_\beta, \Delta_\gamma)$, aside from the layer-depth p. It is in the determination of the optimal values of these two parameters that the two variants of the LR-QAOA algorithm differ.

5.3.1 Normalized Hamiltonian LR-QAOA

Like the standard QAOA, this method also uses an Ising Hamiltonian formulation of the QUBO at hand, with one key change that the Hamiltonian is normalized with the absolute maximum of the Ising coupling term W_{ij} (off-diagonal entry in the Ising Hamiltonian H_F in Equation (4.34)). By noting (using Equation (4.40)) that

$$U_F(\gamma) = e^{-i\gamma \frac{H_F}{\lambda}} = e^{-i\frac{\gamma}{\lambda} H_F},$$

where $\lambda = 1/2 \max |W_{ij}|$ is the absolute maximum of the Ising coupling terms W_{ij}, and that all γ-values by definition of the LR-QAOA ansatz (Equation (4.48)) are proportional to (Δ_γ), the same effective Hamiltonian normalization can be achieved by scaling Δ_γ by λ. In this work, we followed the latter procedure for the Hamiltonian normalization (as opposed to directly normalizing the Hamiltonian, followed in the original paper). The LR-QAOA parameter obtained from a non-normalized Hamiltonian $\Delta_\gamma^{\text{opt}}$ is related to that obtained from a normalized Hamiltonian $\Delta_\gamma^{(\text{opt, norm})}$ as:

$$\Delta_\gamma^{(\text{opt, norm})} = \frac{\Delta_\gamma^{\text{opt}}}{\lambda}. \tag{5.2}$$

The 2D performance landscape for a small problem ($N = 10$) was plotted (see Figure 5.7), which was then used to determine the optimal values of the Δ_γ and Δ_β parameters (a performance landscape [18] is a plot of the probability of the optimal bitstring over a range of Δ_γ and Δ_β parameters, which is obtained by evaluating the QAOA circuit for different values of these parameters). The optimal coordinates $(\Delta_\gamma^{\text{opt}}, \Delta_\beta^{\text{opt}})$ were then determined by finding the point $(\Delta_\gamma, \Delta_\beta)$ in the performance landscape that corresponds to the maximum probability (Equation (4.45)).

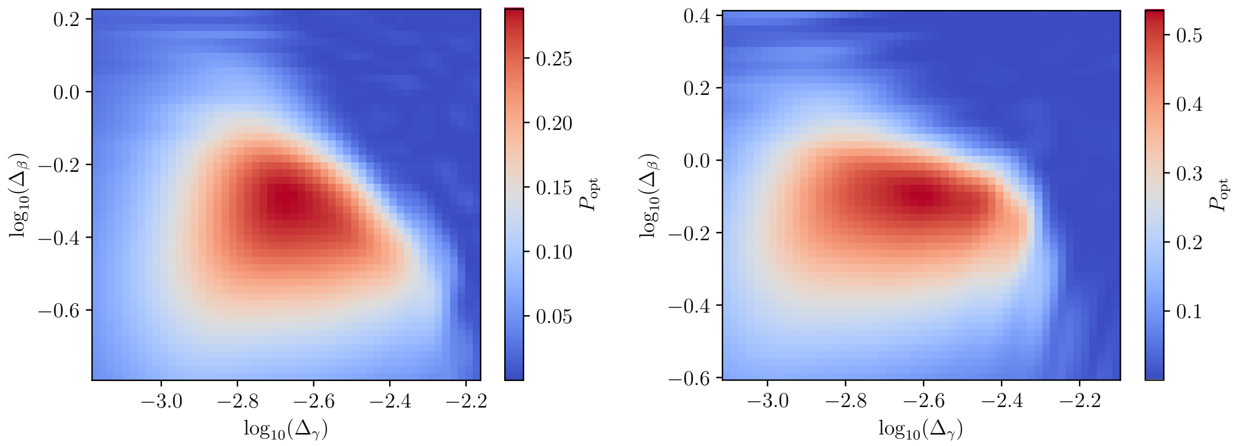
Once the $\Delta_\beta^{\text{opt}}$ and $\Delta_\gamma^{\text{opt}}$ were obtained, fixing the iteration depth of the QAOA at $p = N$ for each instance, the γ and β ramp was set up as in Equation (4.48) Further, the QAOA circuit was constructed (Figure 4.10) and executed.

According to the original paper [19], the normalization procedure yielded the performance landscapes having optima within the same region, agnostic of the problem studied. As a result, the authors fix the Δ_γ and Δ_β parameters throughout all the experiments to the values $\Delta_\gamma^{(\text{norm})} = 0.6$ and $\Delta_\beta^{(\text{norm})} = 0.3$ for all the instances. This is a significant difference from the extrapolation-based LR-QAOA algorithm, which tunes the Δ_γ and Δ_β parameters for each problem instance separately. In the experiments performed in this work, after the construction of the performance landscape for a small problem ($N = 12$), the Δ_γ values for each instance were calculated as described above (with constant $\Delta_\gamma^{(\text{opt, norm})}$), while the Δ_β value was kept as the same (from the performance landscape of the smaller instance) for all larger problems.

5.3.2 Extrapolation-based LR-QAOA

The extrapolation-based LR-QAOA algorithm builds on the experimental observations that the optimal values of the Δ_γ and Δ_β parameters for increasing problem sizes follow a linear trend on a log-log scale, which enables the extrapolation of the optimal parameters for larger problem sizes based on the values obtained for smaller problem sizes. These parameters obtained after extrapolation are then used to construct the QAOA circuit for the larger problem, which is further evaluated on a quantum hardware (or simulator, as in our case).

The given steps were followed to implement the extrapolation-based LR-QAOA algorithm



(a) (b)

Figure 5.7: Performance landscapes plotted as a heatmap on a log-log scale for a size $N = 10$ instance for (a) weighted complete graph, $w_{ij} \in \mathbb{Z}$ and $w_{ij} \in [1, 1000]$ with QAOA circuit of $p = 10$, with optimum at around $(\Delta_\gamma^{\text{opt}}, \Delta_\beta^{\text{opt}}) \approx (0.002, 0.521)$, (b) weighted R3R graph with $w_{ij} \in \mathbb{R}$ and $w_{ij} \in [1, 1000]$ of the same N and p values, with optimum at around $(\Delta_\gamma^{\text{opt}}, \Delta_\beta^{\text{opt}}) \approx (0.002, 0.800)$.

in this work, as described in Reference [20].

1. For a given problem instance (formulated as a QUBO of size N), 10 random sub-samples of the problem were generated by randomly selecting $\tilde{N}$ variables from the original problem, where $\tilde{N} < N$. Since in all the experiments, we looked at problem sizes in the range $[12, 28]$, we fix $\tilde{N}$ to be $\{4, 6, 8, 10\}$ bits.

   This sub-sampling needs to be done in a way that the resulting QUBO samples are representative of the original problem. For the weighted complete MaxCut problem, we did the sub-sampling for each matrix as follows: choosing $\tilde{N}$ random indices from the QUBO matrix of size N, we extracted the corresponding rows and columns from the QUBO matrix to obtain a sub-matrix of size $\tilde{N}$. At each row of the sub-matrix, we replaced the diagonal entry with the sum of the off-diagonal entries. This created QUBO matrices which are similar in structure to that obtained by converting the adjacency matrix of a MaxCut instance to a QUBO matrix (Equation (2.7)). However, this procedure cannot be translated to the weighted R3R graphs, as the sub-sampling following this method may create invalid graphs with one or multiple isolated nodes, which is not representative of the main problem anymore. Therefore, we restricted the study of the extrapolation method to weighted complete graphs for the MaxCut problem.

   For the TA problem, the data set was created with a symmetric distribution of technicians T and assets A at each N to enable this sub-sampling procedure. As explained in section Section 5.2.2, the instance sizes N are set to be even numbers in the range of $[12, 28]$, with $T = A = N/2$ at each N, and the constraint factors $T_1 = A_1 = \lfloor N/4 \rfloor$.

The sub-sampling is then done by randomly selecting $\tilde{T}$ technicians and $\tilde{A}$ assets from the distance matrix at that main problem instance, and extracting the corresponding rows and columns to obtain a sub-matrix (note that this sub-matrix is not a QUBO yet, since it does not contain constraints) of size $\tilde{N} = \tilde{T} + \tilde{A}$. Using this distance matrix, we generate a QUBO for the sub-problem by following the same procedure as in Section 2.3.3 (see Equation (2.16)).

2. Choosing an instance of the smallest sub-problem of size $\tilde{N} = 4$, we generated the performance landscape to obtain the optimal values of the Δ_γ and Δ_β parameters for this sub-problem. The diagram was plotted as a grid of 11×11 points in the $\log(\Delta_\gamma)$ - $\log(\Delta_\beta)$ parameter space (see Figure 5.8 (a)), fixing p to be a fixed value of p_0. This was repeated for all 10 sub-problem instances of size $\tilde{N} = 4$, and the average success probability is calculated as $\bar{P}_{\text{opt}}$.

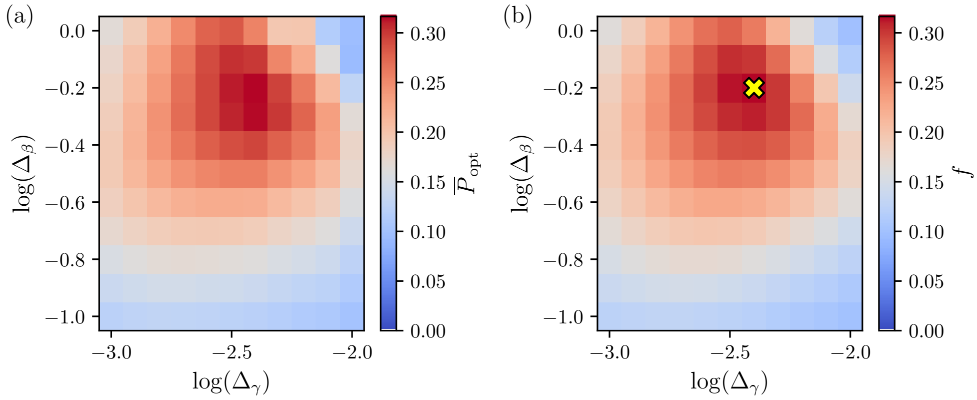


Figure 5.8: (a) Performance landscape for $\tilde{N} = 4$ sub-problem (weighted complete MaxCut) with $p_0 = 6$, plotted as a heatmap on a log-log scale. (b) The grid data fitted to a skewed Gaussian function, with the cross indicating the optimal point $(\Delta_\gamma^{\text{opt}}, \Delta_\beta^{\text{opt}})$.

3. The profile of the region around the optimum was then fitted to a skewed Gaussian function of the form

$$\begin{aligned} f_{\boldsymbol{\mu},\Sigma,\boldsymbol{\alpha},A,B}(\mathbf{x}) = A \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \\ \times \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{\boldsymbol{\alpha}\cdot(\mathbf{x}-\boldsymbol{\mu})}{\sqrt{2}}\right)\right] + B, \end{aligned} \tag{5.3}$$

where $\mathbf{x} = (\log\Delta_\gamma, \log\Delta_\beta)$, and the fit parameters are $\boldsymbol{\mu}$, a two-dimensional vector which denotes the center of the Gaussian, Σ^{-1} (of shape 2×2) the inverse covariance matrix, $\boldsymbol{\alpha}$ a two-dimensional vector representing the skewness of the Gaussian, and $A, B \geq 0$. The fitted function is then used to determine the optimal point $(\Delta_\gamma^{\text{opt}}, \Delta_\beta^{\text{opt}})$ numerically by finding the maximum of the function $f_{\boldsymbol{\mu},\Sigma,\boldsymbol{\alpha},A,B}(\mathbf{x})$. The value of the optimum, along with the widths of the best-fit gaussian profile $s_\gamma = \sqrt{\Sigma_{1,1}}$ and $s_\beta =$

$\sqrt{\Sigma_{2,2}}$, are then stored for the extrapolation step. This fitting procedure is done once for each subproblem size $\tilde{N}$, and the average success probability $\bar{P}_{\text{opt}}$ is also stored for each sub-problem size. The fitted profile for the sub-problem size $\tilde{N} = 4$ is shown in Figure 5.8 (b), where the cross indicates the optimal point $(\Delta_\gamma^{\text{opt}}, \Delta_\beta^{\text{opt}})$.

4. Using the best fit parameters for the sub-problem size $\tilde{N} = 4$, the grid search was set up for the next sub-problem size $\tilde{N} = 6$, by searching in a window defined by

$$\begin{aligned}
\log(\Delta_\gamma)_{\text{min}} &= \log(\Delta_{\gamma,\text{opt}}) - s_\gamma, \\
\log(\Delta_\gamma)_{\text{max}} &= \log(\Delta_{\gamma,\text{opt}}) + s_\gamma, \\
\log(\Delta_\beta)_{\text{min}} &= \log(\Delta_{\beta,\text{opt}}) - s_\beta, \\
\log(\Delta_\beta)_{\text{max}} &= \log(\Delta_{\beta,\text{opt}}) + s_\beta.
\end{aligned} \tag{5.4}$$

where s_γ and s_β are the widths of the best-fit Gaussian profile for the sub-problem size $\tilde{N} = 4$. Steps 2 and 3 were then repeated, i.e., grid-search followed by skewed-gaussian fit of the average grid, to obtain the optimal values of Δ_γ and Δ_β for the sub-problem size $\tilde{N} = 6$, and the process was continued for sub-problem sizes up to $\tilde{N} = 10$. An example plot of these parameters is shown in Figure 5.9 for the MaxCut problem, where the parameters Δ_γ and Δ_β are plotted as a function of the sub-problem size $\tilde{N}$, along with the average success probability $\bar{P}_{\text{opt}}$ obtained from the grid search.

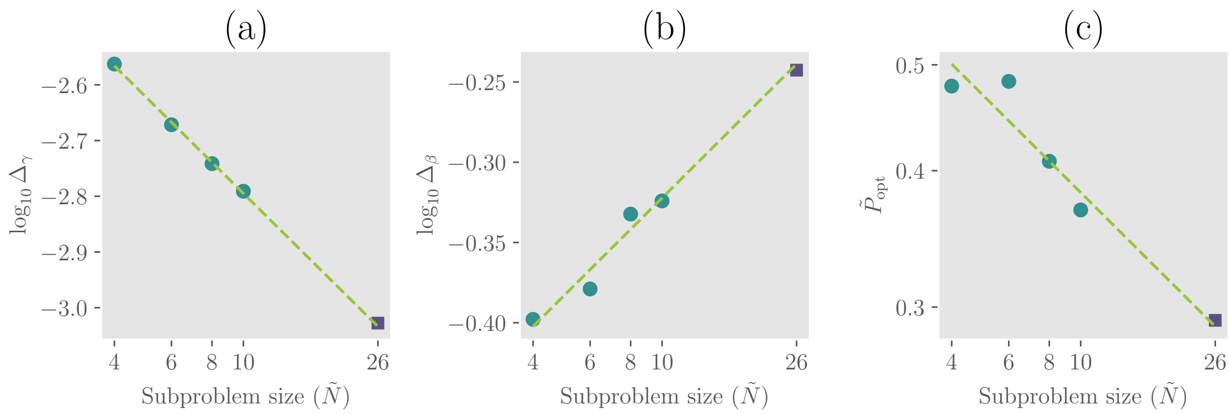


Figure 5.9: Extrapolation of LR-QAOA parameters for the weighted complete MaxCut problem. The parameters Δ_γ (a) and Δ_β (b) are plotted as a function of the sub-problem size $\tilde{N}$, along with (c) the average success probability $\bar{P}_{\text{opt}}$ obtained from the grid search. The dashed lines are linear fits (on the log-log scale) to the data points, which are used to extrapolate the parameters for larger problem sizes. Note that in this plot the extrapolation is only representative since the data points are averaged over the 10 sub-problem instances.

5. As seen in Figure 5.9, the parameters Δ_γ, Δ_β and P_{opt} are found to follow a linear trend on a log-log scale, which is fitted to a linear function. Extrapolating the linear

fit to larger problem sizes, we obtain the optimal values of Δ_γ and Δ_β for the original problem size N.

6. The extrapolated parameters are then used to construct the QAOA circuit for the original problem size N, which is further evaluated on a quantum hardware (or simulator, as in our case). In order to do this, we still need the optimal value of the layer depth p for the QAOA circuit. To obtain this, we look at Equation (5.1) and set N_{shots} to be the number of executions for a 50% success probability of finding the optimal solution, which is obtained as

$$N_{\text{shots}} = \frac{\log\left(1/2\right)}{\log\left(1 - P_{\text{opt}}\right)}, \tag{5.5}$$

where P_{opt} is the probability determined by extrapolation in step 5. The optimal p was then determined by minimizing D in Equation (5.1) with respect to p, i.e., starting from an initial p value, we altered the value of p until the total depth D is minimized. The p values were iterated from a logarithmically spaced set of values. The final values of Δ_γ, Δ_β, and p are then used to construct the QAOA circuit, which is further executed.

5.4 Results from Quantum Algorithms

In this section, we present the results of the quantum algorithms implemented for the MaxCut and Technician-Asset Allocation (TA) problems. The results are presented in two parts: first, we discuss the results of the two LR-QAOA algorithm runs for both problems, and then we compare the performance of the LR-QAOA algorithm with the classical algorithms discussed in Section 5.2

5.4.1 Results from MaxCut Instances

Weighted Complete Graphs

The linear ramp parameters Δ_γ and Δ_β which were obtained from extrapolation for the weighted complete MaxCut problem are shown in Figure 5.10. The parameter $\log_{10}\Delta_\gamma$ shows a steady decrease with increasing problem size N, while on the other hand, $\log_{10}\Delta_\beta$ does not exhibit any clear trend. The probabilities (to measure the optimal bitstring obtained classically) on evaluating the circuit, while showing increasing variation from the predicted probabilities (from extrapolation), are found to give an average of about 10-20% for all problem sizes N (see Figure 5.11 (a)). The difference between the extrapolated and measured probabilities is also shown in Figure 5.11(b), which shows that the difference increases with increasing N, indicating that the extrapolation method could be improved further to obtain better probabilities for larger problem sizes.

The parameters $\log_{10}\Delta_\gamma$ used to evaluate the normalized Hamiltonian LR-QAOA algorithm (obtained as explained in Section 5.3.1) are also shown in Figure 5.12 (a) for comparison. The extrapolated parameters are found to be in the same order of magnitude as the normalized Hamiltonian parameters, indicating that the extrapolation method is able to capture the trend of the optimal parameters for the MaxCut problem. Additionally, the difference between the extrapolated and normalized Hamiltonian parameters is shown in Figure 5.12 (b), which shows that the difference increases with increasing N, similar to the trend observed in Figure 5.11(b). An interesting observation is that the highest average probabilities obtained from the extrapolation-based LR-QAOA (Figure 5.11 algorithm are found at $N = 17$, 20 and 24, which is in the same range as the range where the difference in $\log_{10} \Delta_\gamma$ is closest to zero. The measured probabilites from the normalized Hamiltonian LR-QAOA algorithm are shown in Figure 5.13, which shows that the average probability is also around 10% for all problem sizes N, similar to (or slightly larger than) the extrapolation-based LR-QAOA algorithm.

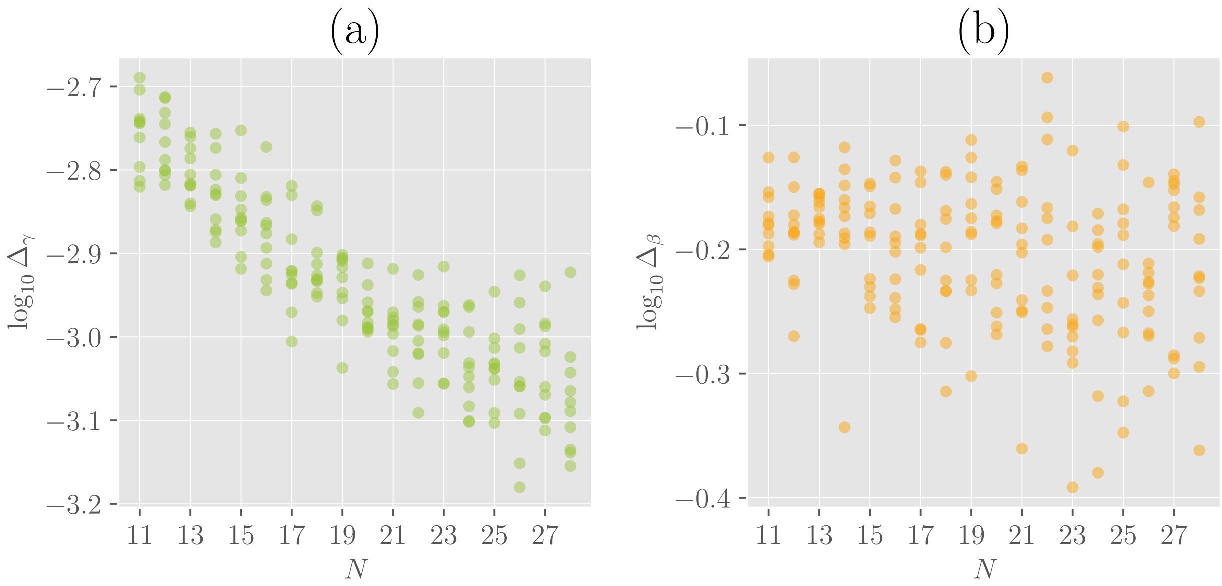


Figure 5.10: Extrapolated LR-QAOA parameters for the MaxCut problem on weighted complete graphs. The parameters $\log_{10} \Delta_\gamma$ (a) and $\log_{10} \Delta_\beta$ (b) are scattered as a function of the problem size N.

Step-ups in the measured probabilities were observed when the layer depth p was increased to multiples of N, i.e., $p = 2N, 3N, 4N$ for the normalized Hamiltonian LR-QAOA algorithm. The measured probabilities for these layer depths are shown in Figure 5.14, which shows that the average probability for the smallest instance went up to around 70% for $p = 4N$ and $N = 11$, and around 40% for the largest instance $N = 28$. This indicates that the probabilities are improving with increasing layer depth, and that the normalized Hamiltonian LR-QAOA algorithm is able to capture the structure of the MaxCut problem better with larger p values. As QAOA can be seen as a Trotterized version of the adiabatic annealing algorithm, this improvement in probabilities with increasing p is expected, as the QAOA circuit is able to explore the solution space more effectively with larger layer depths. However, it is important to note that the total depth of the circuit also increases with increasing p, which may lead to longer runtimes on quantum hardware, making it more

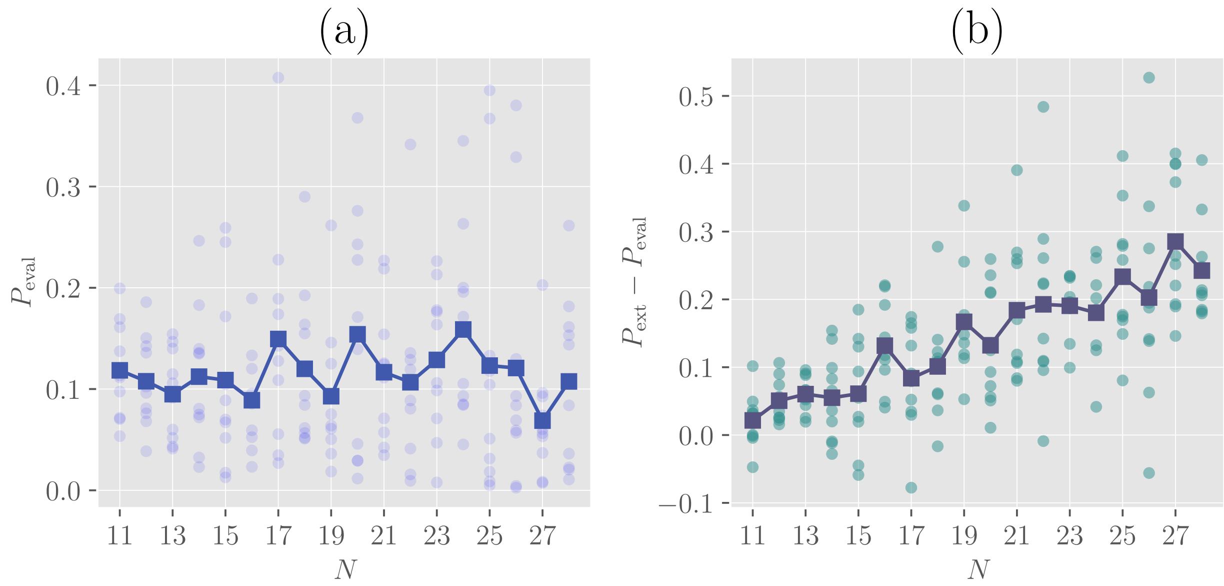


Figure 5.11: (a) Measured probabilities P_{eval} for the MaxCut problem on weighted complete graphs, using extrapolated LR-QAOA parameters. The probabilities are scattered (dots) as a function of the problem size N, with the mean probability across 10 instances (square markers) shown at each N. On evaluation, an average probability of about 10% is obtained at each N. (b) Difference between the estimated probabilities from extrapolation and the measured probabilities, plotted as a function of N. The difference is found to increase with increasing N, indicating a potential for improvement in the extrapolation of probabilities.

susceptible to noise and decoherence effects.

The total depths of the circuits (which is an indicator of the runtime of the LR-QAOA algorithm) for both the extrapolation-based and normalized Hamiltonian LR-QAOA algorithms are shown in Figure 5.15. The depths D scale better with increasing problem size N for the normalized Hamiltonian LR-QAOA algorithm compared to the extrapolation-based LR-QAOA algorithm. The interesting observation is that the former attains similar performance (indicated by probabilities in Figure 5.13) with lower p values (see Figure 5.12 (c)), which in turn leads to lower total depths D for the normalized Hamiltonian LR-QAOA algorithm. The largest circuit depth in the entire dataset was found to be around $D \approx 10^5$ for the normalization based LR-QAOA algorithm, while the extrapolation-based LR-QAOA algorithm had multiple circuits with depths one order-of-magnitude larger than that, with the largest depth being around $D \approx 10^6$.

Weighted R3R Graphs

As explained before, the extrapolation-based LR-QAOA algorithm was not used for the weighted R3R graphs, as the sub-sampling procedure described in Section 5.3.2 does not work for these graphs. We set up instances with even sizes $N = 12, 14, \ldots, 28$ to satisfy the condition for valid R3R graphs (the product of degree and N should be even: since degree

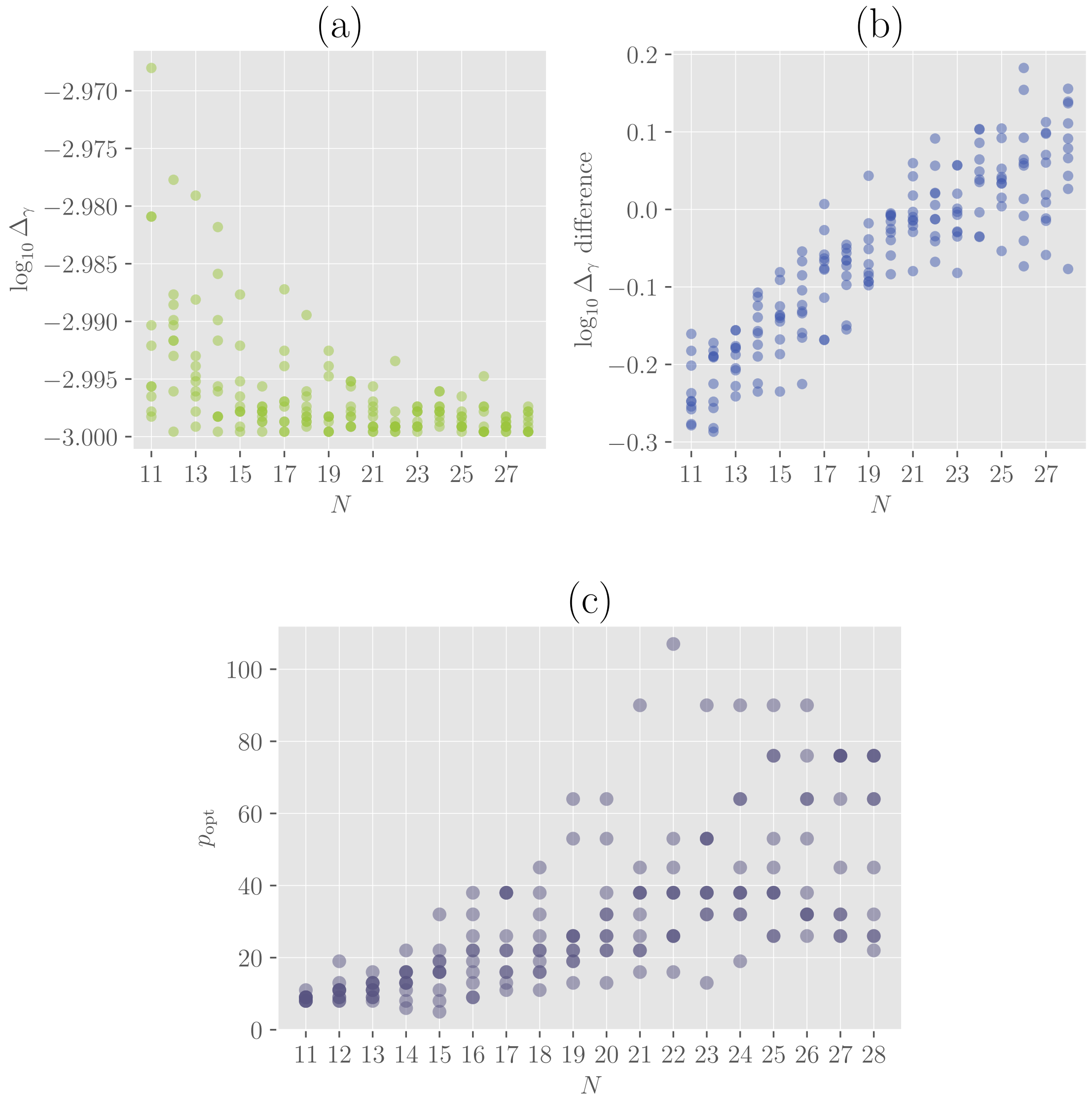


Figure 5.12: Comparison of $\log_{10} \Delta_\gamma$ in normalization-based LR-QAOA (a), its difference from extrapolation-based LR-QAOA (b), and optimal depth p from extrapolation (c) for weighted complete MaxCut. In (a), $\log_{10} \Delta_\gamma$ stabilizes near -3 as N increases; (b) shows a growing difference with N.

is fixed to be 3, the number of nodes should be even; see Section 2.3.2).

As in the weighted complete graphs, the Δ_β parameter was obtained from the performance landscape for a small instance (see Figure 5.7b), and Δ_γ determined at runtime for each instance. The calculated Δ_γ is shown in Figure 5.16, which shows a similar trend as in Figure 5.10 (a), with the parameter $\log_{10} \Delta_\gamma$ taking similar values. This parameter, whose value is related to the maximum value in the QUBO matrix, is expected to be similar for both weighted complete and R3R graphs, since the weights are sampled from the same distribution $[1, 1000]$.

The measured probabilities from the normalized Hamiltonian LR-QAOA algorithm evaluated with $p = N$ are shown in Figure 5.17a, which shows that the average probability is

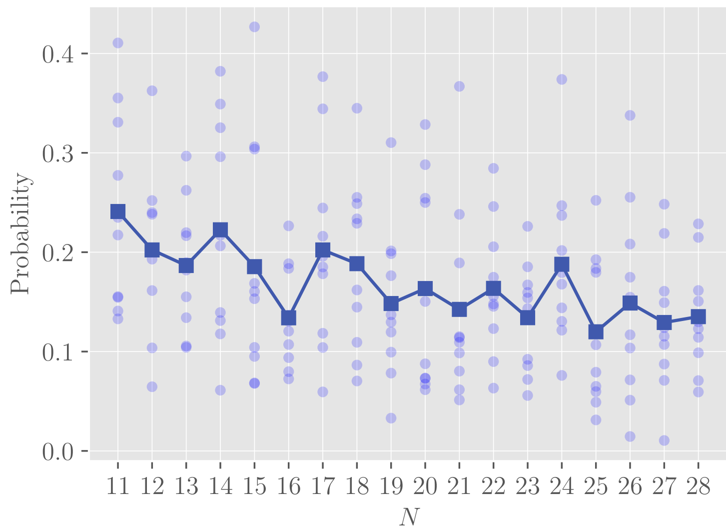


Figure 5.13: Average measured LR-QAOA probabilities for the MaxCut problem on weighted complete graphs, using normalized Hamiltonian LR-QAOA parameters, with $p = N$. The probabilities are scattered (dots) as a function of the problem size N, with the mean probability across 10 instances (square markers) shown at each N. The average probability is found to be around 10% for all problem sizes N, similar to the extrapolation-based LR-QAOA algorithm.

around 10% for most problem sizes N. However, for the larger instances ($N \geq 26$), the probabilities drop to around 5%. The corresponding total depths D for these instances are shown in Figure 5.17b. Owing to the dip in probability for the larger instances, the total

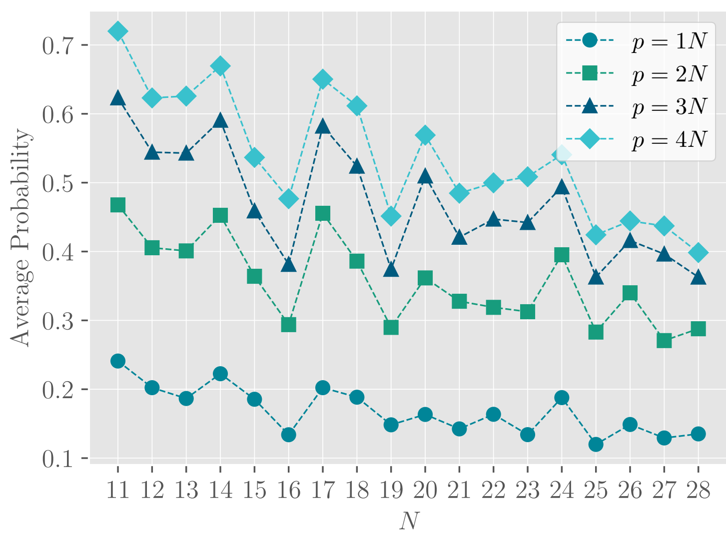


Figure 5.14: Measured LR-QAOA probabilities for the MaxCut problem on weighted complete graphs, using normalized Hamiltonian LR-QAOA parameters. The results show that increasing the layer depth p to multiples of N (e.g., $p = 2N, 3N, 4N$) leads to a slight improvement in the measured probabilities.

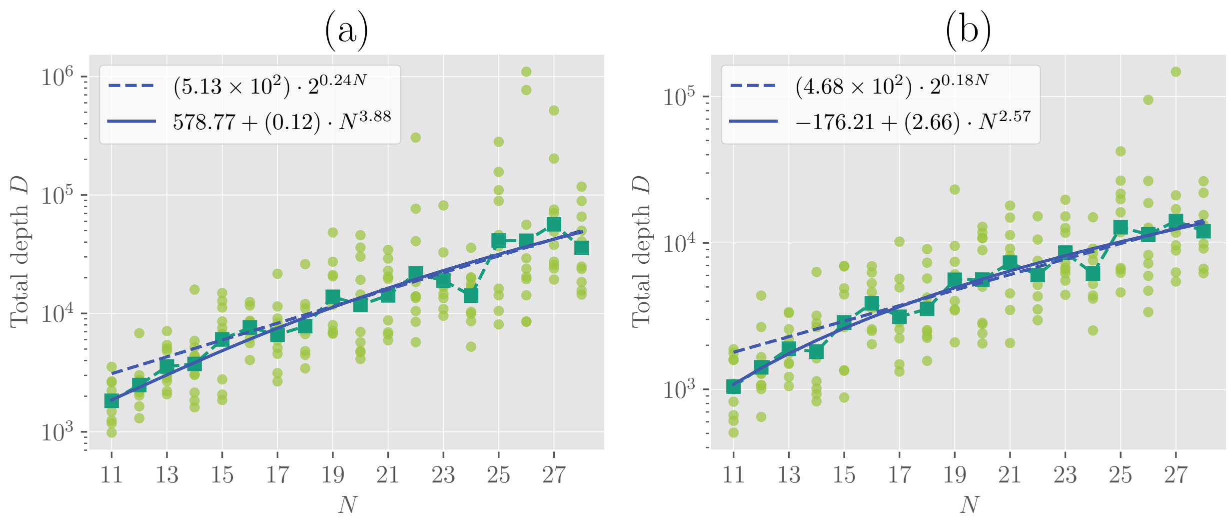


Figure 5.15: Scaling of total circuit depths for finding the optimal solution of the MaxCut problem on weighted complete graphs, for both extrapolation-based (a) and normalized Hamiltonian with $p = N$ (b) LR-QAOA algorithms. The total depths of all instances (dots) are plotted as a function of the problem size N, with the geometrically averaged depth across 10 instances (squares) shown at each N. The data fits well to both an exponential (dashed lines) and a polynomial (solid lines) function. From both fits, it can be seen that the normalized Hamiltonian LR-QAOA algorithm has a lower scaling of depths compared to the extrapolation-based LR-QAOA algorithm.

depths also show a similar trend, with the depths taking a spike at $N = 26$, which could be attributed to a larger number of shots required to find the optimal solution compared to the smaller instances.

5.4.2 Results from Technician-Asset Allocation Instances

We tested both LR-QAOA methods on the Technician-Asset Allocation (TA) problem, with 10 instances each of sizes $N = 12, 14, \ldots, 28$, with $T = A = N/2$ technicians and assets, and $T_1 = A_1 = \lfloor N/4 \rfloor$ constraint factors. The parameters resulting from the extrapolation-based LR-QAOA algorithm are shown in Figure 5.18, which shows a similar behaviour as for the MaxCut instances (Figure 5.10): the parameter $\log_{10} \Delta_\gamma$ shows a steady decrease with increasing problem size N, while $\log_{10} \Delta_\beta$ does not exhibit any clear trend. While the predicted probabilites from extrapolation showed promise of around 20% for the TA problem (Figure 5.19 (a)), the evaluated probabilities from the extrapolated parameters drastically differ from these extrapolated values. The actual performance decline rapidly, from approximately 8% in the best case at the smallest instance size, to essentially zero for larger instances ($N \geq 16$). This large discrepancy between predicted and measured probabilities suggests that the extrapolation method does not reliably capture the complexity of the TA optimization landscape.

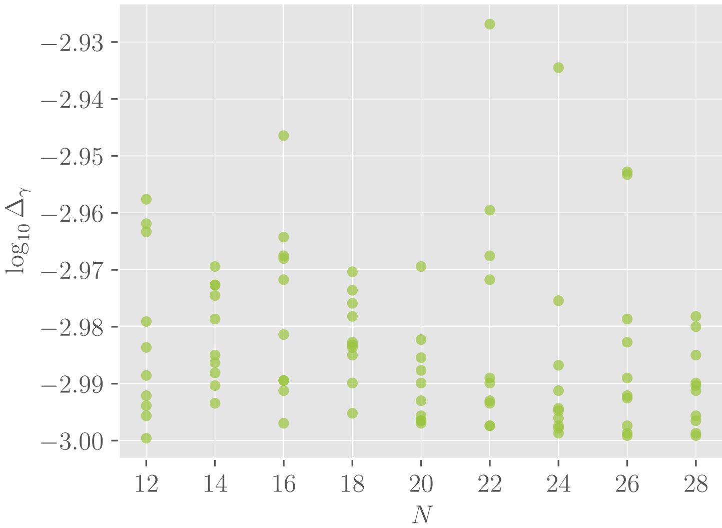
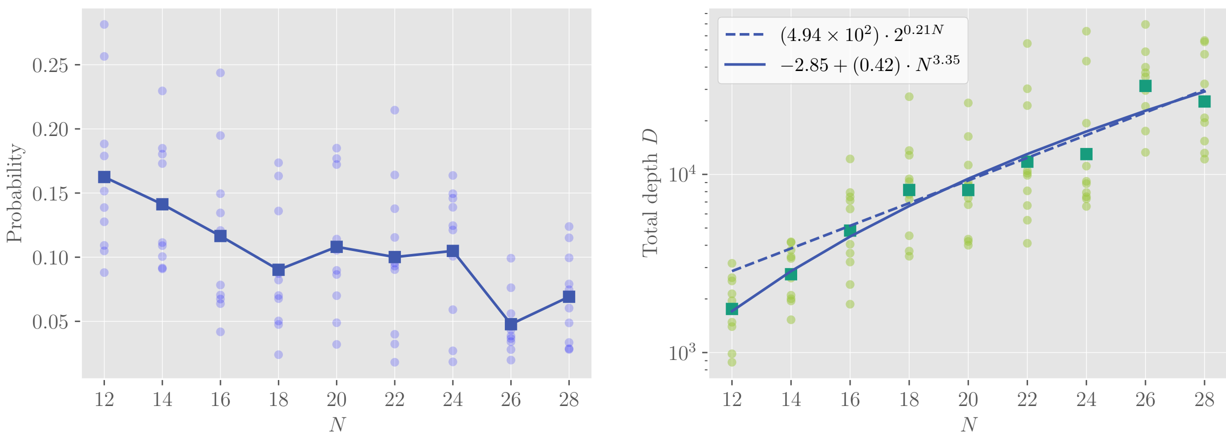


Figure 5.16: LR-QAOA parameter $\log_{10}\Delta_\gamma$ for the MaxCut problem on weighted R3R graphs. The range of values follow a similar pattern as in Figure 5.12 (a), since the weights of the R3R graphs are also sampled from the same distribution $[1, 1000]$.



(a) (b)

Figure 5.17: (a) Measured LR-QAOA probabilities for the MaxCut problem on weighted R3R graphs, using normalized-Hamiltonian LR-QAOA parameters, evaluated with $p = N$. The average probabilities (squares) remain close to 10% for most problem sizes N, with the exception of the $N \geq 26$ instances, where they drop to around 5%. (b) Scaling of total circuit depth for the same instances: an exponential fit yields a scaling factor of 0.21, while a polynomial fit gives 3.35.

Further insight into the limitations of the extrapolation-based approach is seen by analyzing the total circuit depths, which scales with problem size as:

$$D(N) \approx 10.22 \times 2^{1.02N},$$

for the extrapolation-based LR-QAOA algorithm, which is shown in Figure 5.21 (a) indicating a purely exponential behaviour (i.e., growing as 2^N with no fractional exponents). Such a scaling of the depth is not better than a direct brute-force search evaluating all 2^N bitstrings. This rapid growth of total depth is primarily driven by the large values of the optimal layer count p, with the extrapolation method suggesting $p > 100$ layers for multiple instances of larger sizes, and even exceeding $p = 250$ for a specific instance (see Figure 5.18 (c)). Consequently, the total circuit depths D escalated quickly from already big values (approximately 10^5 for $N = 12$) to unrealistic scales (around 10^{11} for $N = 28$). Despite constructing such deep circuits, the measured success probabilities remain meager.

In the case of the normalized Hamiltonian based approach, the $\log_{10} \Delta_\gamma$ parameter was at a constant value, since the maximum value in the QUBO matrix was capped at the constraint value $2D_{\text{lim}}$. Since the original method already fixes $\log_{10} \Delta_\beta$, this means that the LR-QAOA parameters for the normalized Hamiltonian method were fixed for all instances. The evaluated probabilities from the normalized Hamiltonian LR-QAOA algorithm are shown in Figure 5.20. Despite evaluating the circuit with increasing p values in multiples of N, no improvement in the probability was observed, where the best probability observed was less than 1% for $p = N$. The corresponding depths (for $p = N$) scale only slightly better than that in the case of the extrapolation method (Figure 5.21):

$$D(N) \approx (4.5 \times 10^2) \times 2^{0.84N},$$

with the largest depth observed in this method being one order of magnitude smaller than that of the former. However, since the measured probability remained under 1%, the normalized Hamiltonian LR-QAOA method also did not yield promising results for the TA problem. These results from the quantum algorithms are strikingly different from those obtained from the classical algorithms (Section 5.2.2, Figure 5.6), where the runtime scaled with an exponential factor of 0.07 and 0.12 for the two classical algorithms, and also obtained optimal solutions for all instances in the set.

5.5 Comparison of Classical and Quantum Scaling

In this final section, we conclude our study by comparing the scaling of the LR-QAOA methods with the classical algorithms discussed in Section 5.2. To make a consistent comparison, we run the classical algorithms on the same instances used for the LR-QAOA methods, and compare the scaling of the runtime in this particular problem size range. Additionally, while

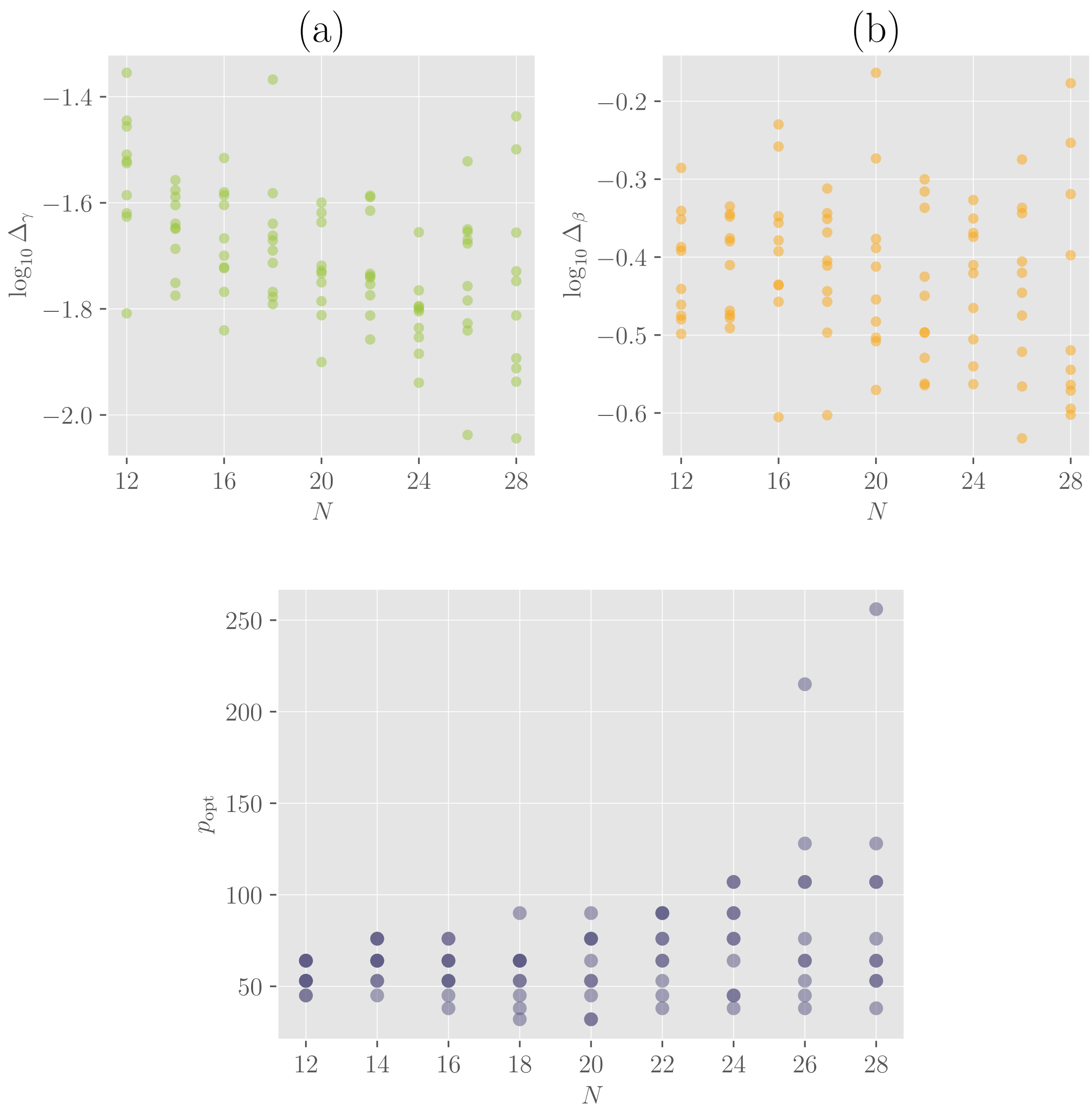


Figure 5.18: Extrapolated LR-QAOA parameters for the TA problem, similar to Figure 5.10 for the MaxCut problem. The parameters $\log_{10}\Delta_\gamma$ (a) and $\log_{10}\Delta_\beta$ (b) and optimal layer depth $p_{\rm opt}$ are scattered as a function of the problem size N. The $\log_{10}\Delta_\gamma$ parameter shows a steady decrease with increasing N, while $\log_{10}\Delta_\beta$, again does not exhibit any clear trend.

a polynomial fit also works for most of the scaling plots, we set the exponential function

$$T(N) \approx a \times 2^{bN}, \tag{5.6}$$

where a and b are the fitting parameters (b is the scaling factor), to compare the scaling of all algorithms. This choice is made on the basis of the suitability of the function's fit to all algorithms.

This comparison provides insight into the potential advantages and limitations of the LR-QAOA methods in solving the MaxCut and TA problems. Since the LR-QAOA methods did not result in promising solutions for the TA problem, we focus on the MaxCut problem

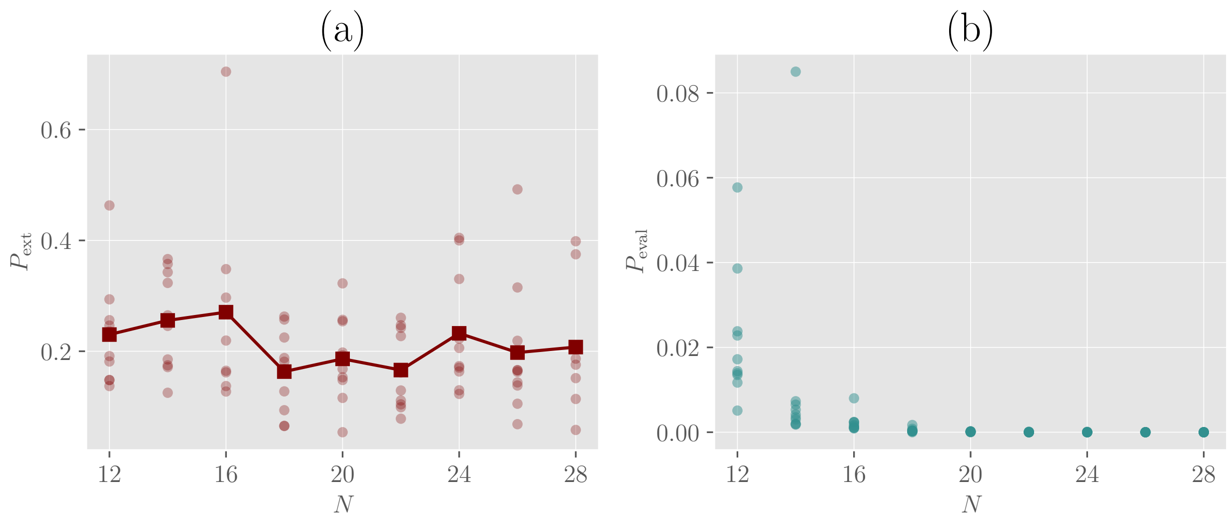


Figure 5.19: (a) Extrapolated LR-QAOA probabilities P_{ext} for the TA problem, plotted as a function of the problem size N. The predicted trend shows a mean around 20% across all instances. (b) Evaluated LR-QAOA probabilities P_{eval} for the TA problem, plotted as a function of the problem size N. The measured probabilities show that the circuit evaluation with the extrapolated parameters did not have the desired result, with probabilties being close to zero even for the smallest instances in the set.

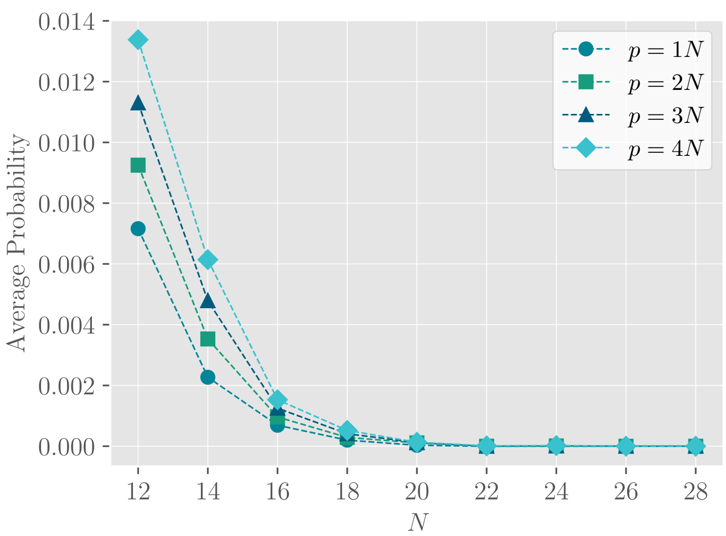


Figure 5.20: Measured LR-QAOA probabilities for the TA problem (averaged over 10 instances), using normalized Hamiltonian LR-QAOA parameters. The QAOA circuit was evaluated with $p = N, 2N, 3N$ and $4N$, where in all cases, the measured probability did not indicate significant improvement, remaining below 1% for all problem sizes N.

for this comparison.

In the range of the problems studied, we first identify the solver which offers the best performance for the MaxCut problem, in terms of scaling and quality of the corresponding

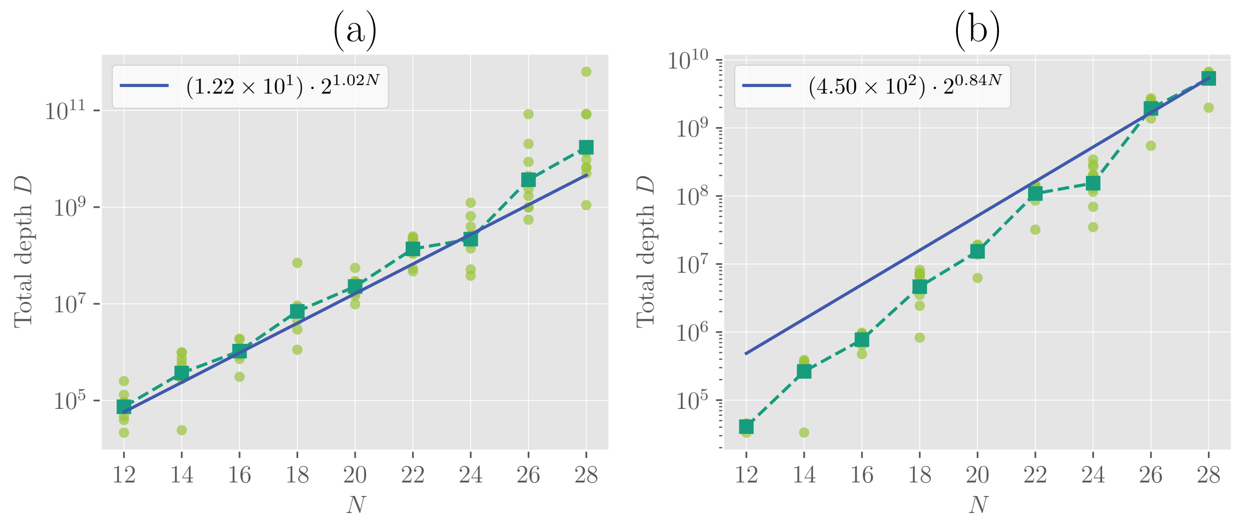


Figure 5.21: Same as Figure 5.15 for the TA problem. The behaviour indicates depths scaling by five to six orders of magnitude for both methods, indicating poor scaling of the LR-QAOA methods for the TA problem.

solution. CPLEX shows an unrealistic scaling behaviour in this range of problems, where the average runtime across the 10 instances shows a slight *decrease* with increasing problem size N, according to the exponential fit

$$T(N) \approx 97.52 \times 2^{-0.03}N,$$

which is shown in Figure 5.22 (a). However, this is likely due to the fact that, in this regime of N values, the CPLEX pre-solve works quite efficiently for the specific problem instances, finding symmetries and reducing the problem size significantly. This is also evident from the spread of runtimes for some of the larger problem instances, where the fastest runtime is around 30 seconds, while the slowest runtime is over 100 seconds (see $N = 24$ in Figure 5.22 (b)). Additionally, we also saw in the asymptotic scaling analysis in Section 5.2.1, that the CPLEX solver struggles to solve weighted complete MaxCut problems even for $N = 70$ in a reasonable time, which indicates that the solver's scaling behaviour adapts to the problem size, and that the runtime is not representative of the actual scaling behaviour in this range of N values. However, due to the tolerance factor of CPLEX being set to a low value, the objective value determined by CPLEX is expected to be close to the optimal value. The GW algorithm, on the other hand, shows a more realistic scaling behaviour, with the average runtime across the 10 instances fitting to an exponential function with the smallest overall scaling factor among the three classical solvers used. However, this has to be balanced with the fact that the GW algorithm does not guarantee optimality of the solution, and the objective value determined by this solver is often worse than that of CPLEX as well as MQLib, as seen in Figure 5.22 (d). The quality of solution from GW algorithm scales poorly even when the number of cuts parameter is set to scale linearly with the problem size, with the difference in objective value between the GW and CPLEX solvers becoming larger with increasing N. Hence it is not ideal to include the GW optimizer as the best classical solver in this range, despite its good runtime scaling behaviour.

As a result, in terms of runtime scaling (Figure 5.22 (a)), as well as quality of solution (Figure 5.22 (d)), the `MQLib` solver is found to be the most suitable classical solver for the MaxCut problem in this range of N values. The runtime scales with a rate of 0.18, at the same time, the objective value determined by `MQLib` is found to match with that determined by CPLEX. As seen in the asymptotic scaling analysis, the `MQLib` solver is able to give heuristic solutions to MaxCut problems with N up to 200 in a reasonable time, even though the objective value is not guaranteed to be optimal. This gives yet another reason to choose `MQLib` to be the most suitable solver for the MaxCut problem in this range of N values, and compare the LR-QAOA methods with this scaling factor.

In the case of weighted R3R graph, the GW solver was found to be the one with the poorest scaling in terms of runtime as well as quality of solution (Figure 5.23 (c) and (d)). The `MQLib` solver, on the other hand, shows a good scaling behaviour with a factor of 0.11, and once again finds the same optimal solution as CPLEX, as in the case of the weighted complete graphs. However, since CPLEX does not give a negative scaling behaviour as in the case of weighted complete graphs, we still include it in the comparison with quantum algorithms. However, as previously noted, the small scaling factor in this regime is not expected to hold for larger problem sizes.

The exponents from the fit to the exponential function Equation (5.6) for the classical algorithms, as well as the LR-QAOA methods, are shown in Table 5.1.

For both MaxCut problems, we see that the normalized Hamiltonian LR-QAOA algorithm compares quite well to the classical algorithms. For weighted complete graphs, the normalized Hamiltonian LR-QAOA algorithm has a scaling factor of 0.176, which is slightly better than that of `MQLib` (0.182), while the extrapolation-based LR-QAOA algorithm has a scaling factor of 0.236, which is worse than both `MQLib` and the normalized Hamiltonian LR-QAOA algorithm. The extrapolation-based LR-QAOA algorithm is also found to have a larger total depth, as shown in Figure 5.15, which indicates that, among the two LR-QAOA algorithms, the normalized Hamiltonian approach scales better for the problem instances studied for the weighted complete MaxCut problem. However, it is important to note that the extrapolation-based LR-QAOA algorithm was able to determine the optimal parameters for the MaxCut problem purely heuristically, and further adapts these parameters to each instance, a feature which is lacking in the normalized Hamiltonian LR-QAOA algorithm. This is an important aspect to consider, as the extrapolation-based LR-QAOA algorithm can potentially yield better results for larger problem sizes, where the normalized Hamiltonian LR-QAOA algorithm may not be able to adapt to the specific instance. While the LR-QAOA approaches do not necessarily *outperform* the classical algorithms in terms of runtime scaling, it does show a comparable performance, which is promising.

For the weighted R3R graphs, the normalized Hamiltonian LR-QAOA algorithm has a scaling factor of 0.21, which is slightly worse than that of `MQLib` (0.11). However, with

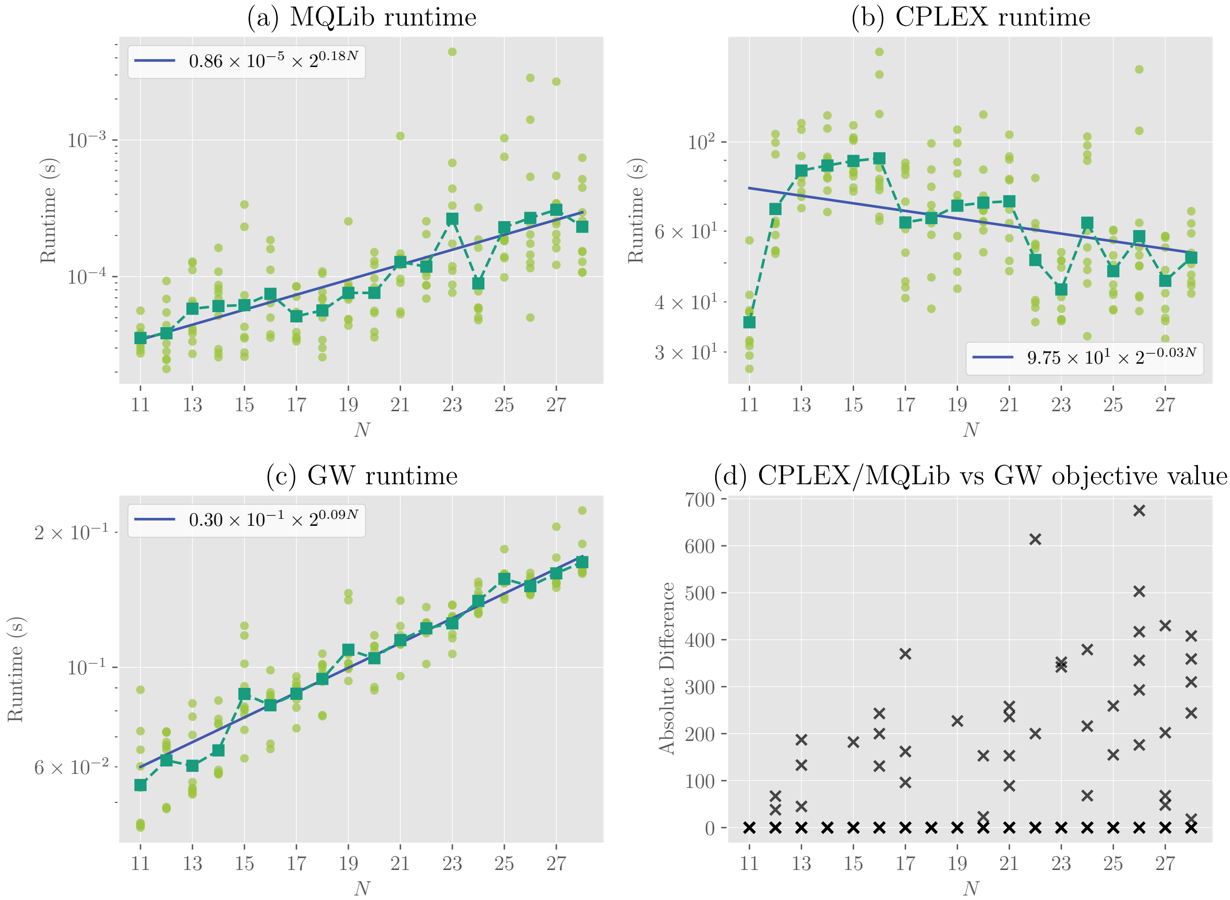


Figure 5.22: (a)-(c) Runtime scaling of classical solvers for the MaxCut problem on weighted complete graphs, (d) optimality of the GW solutions, compared to the CPLEX and `MQLib` solvers, by looking at the absolute difference in objective values.

	Exponents	
	Weighted Complete	Weighted R3R
MQLib	0.182	**0.111**
Extrapolation LR-QAOA	0.236	
NH LR-QAOA		
$p = N$	0.176	0.211
$p = 2N$	**0.173**	0.181

Table 5.1: A comparison of the fitted exponents b for the runtime (Equation (5.6)) of the quantum algorithms with that of the best scaling classical algorithm

the increased iteration depth $p = 2N$ at each N, the normalized Hamiltonian LR-QAOA algorithm is able to achieve a better performance in terms of improved scaling (a feature which we observed for both graph types). However, a further improvement in scaling was

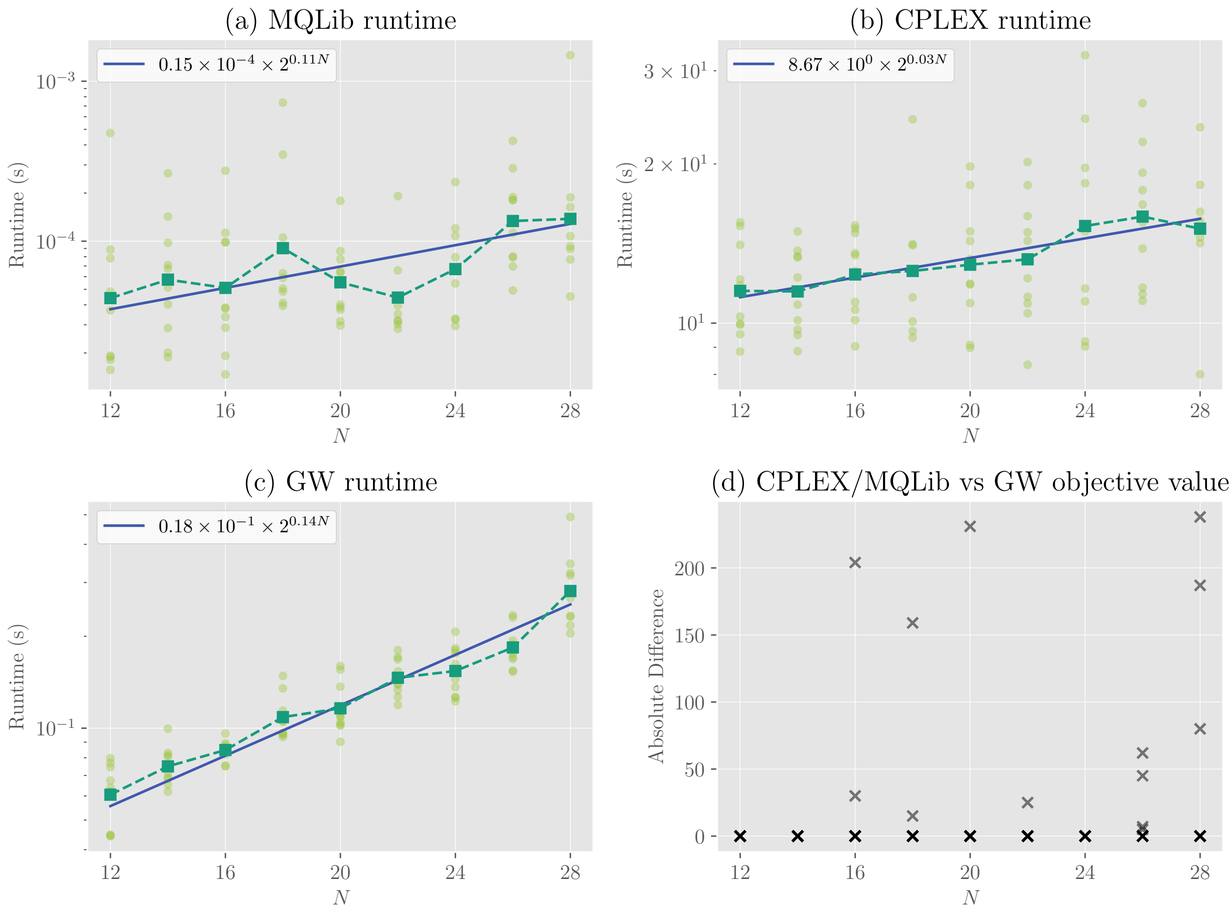


Figure 5.23: Same as Figure 5.22 but for weighted R3R graphs

not observed when setting p to $3N$ or $4N$ (see Appendix C).

In summary, our results show that

- Classical solvers remain efficient in solving the MaxCut problem, with `MQLib` being the overall best classical solver in the problem sizes studied.
- The normalized Hamiltonian LR-QAOA algorithm scales better than the extrapolation method for both MaxCut problems, and both methods scale comparable to the classical algorithms.
- The extrapolation-based LR-QAOA algorithm is able to adapt its parameters to each instance, which can potentially offer better results for larger problem sizes.
- The TA problem did not yield promising results for either quantum algorithms, which indicates scope for further analysis.

Chapter 6

Conclusion and Outlook

In this thesis, we conducted a comprehensive comparison of runtime scaling in classical and quantum optimization methods, focusing on two problems: the MaxCut problem and a new industrial problem which we refer to as the Technician-Asset allocation (TA) problem. MaxCut being a well-known NP-hard problem, acts as a benchmark for optimization algorithms, which is highly prevalent in quantum optimization literature. The TA problem, on the other hand, is a real-world problem that arises in the context of resource allocation in an energy network, where technicians are assigned to assets based on various constraints and preferences. While the MaxCut problem has a well-established formulation, the cost function for the TA problem (encoding the problem constraints and preferences) was derived in this work. The limiting distance between technician-asset pairs for this problem was then set to a common value for all instances, to ensure that the feasibility of the solutions was maintained.

To compare the performance of classical and quantum algorithms, we formulated both problems as Quadratic Unconstrained Binary Optimization (QUBO) problems, which is a common representation for optimization problems in quantum computing, due to its strucutural similarity to the Ising Hamiltonian. A runtime analysis of classical algorithms over a large range of problem sizes ($N \in \{10, 20, \dots, 200\}$ for MaxCut, and $N \in \{4, 6, \dots 50\}$ for the TA problem) was conducted, with the goal of understanding the scaling performance of these algorithms. Later, to enable a comparison with quantum algorithms, we also set up instances for both problems in the range of $N \in \{12, \dots, 28\}$, since this is the range of problem sizes that can be handled by the current quantum simulators.

In the case of quantum algorithms, we implemented two variations of the LR-QAOA algorithm (a modification to the standard and well-studied QAOA algorithm), to test its performance in terms of runtime scaling on the MaxCut and TA problems. The normalized Hamiltonian LR-QAOA, as the name suggests, normalizes the Hamiltonian of the problem, thereby allowing the algorithm to work with a fixed set of parameters, agnostic to the problem instance. The second LR-QAOA algorithm follows a sub-sampling routine to estimate optimal parameters of the problem at hand by generating a set of sub-problems sampled from the main problem, which determines circuit parameters for each instance following a linear extrapolation on a log-log scale.

From an analysis of the runtime of classical algorithms at increasing problem sizes, we found that the `MQLib` solver performed well (in terms of runtime) on both problems, and at both problem ranges studied. Being a heuristic, this method does not give a provable optimality guarantee. Therefore, we confirmed the optimality by comparison of the solutions to more elaborate methods like CPLEX for smaller problem sizes. Although the GW algorithm similarly exhibited fast runtimes as `MQLib` for larger problem sizes, the solution quality of the GW algorithm matched neither that of the CPLEX solver nor that of the `MQLib` heuristics. The `MQLib` heuristics also performed well on the TA problem, again providing optimal solutions for smaller problem sizes, as confirmed by CPLEX for the same instances.

The two quantum algorithms were tested on weighted complete graphs in the case of Max-Cut. Since the sub-sampling routine of the LR-QAOA algorithm could not be applied to the R3R MaxCut instances, we only tested the normalized Hamiltonian LR-QAOA algorithm on these instances. The simulation results showed promising performance for both approaches (albeit not reaching the performance of the classical `MQLib` heuristic), with the normalized Hamiltonian approach offering slightly better scaling performance than the extrapolation method. For the weighted complete MaxCut instances, the scaling of this method shows a slight improvement over the classical algorithms as well as the extrapolation approach, with an additional small scaling improvement when the iteration depth p is increased to $p = 2N$ (over the original $p = N$). The extrapolation approach, while not scaling as well as the normalized Hamiltonian method, still performed comparably to the classical algorithms, and was able to adapt its parameters to each instance, which could potentially offer better results for larger problem sizes.

When testing the TA instances with the LR-QAOA algorithms, we found that the algorithm performance was not satisfactory, with probabilities barely (if ever) exceeding random-guessing levels. The parameters predicted by the sub-sampling routine in the extrapolation method did not show the predicted probabilities on evaluating the circuits. The normalized Hamiltonian approach, which asserts a fixed set of parameters to work across all problem instances, failed to perform as well, despite the iteration depth p being increased to several multiples of N. The total depths of the circuits were significantly larger than those used for the MaxCut problem, despite which the resulting probabilities were still very low.

Our results reaffirm that state-of-the-art classical algorithms remain highly successful in solving the MaxCut problem, even up to moderate problem sizes. The LR-QAOA methods show promise in scaling performance. Nevertheless, as we also saw in our classical asymptotical analysis, an excellent performance of algorithms on small problem sizes does not guarantee a good scaling at larger problem regimes. Furthermore, this performance of the quantum algorithms need not translate equally well to real quantum hardware (as our results are based on idealized, noiseless simulations), and does not account for the noise and errors in current quantum hardware, which become more pronounced with increasing circuit depths.

The significant difference in the performance of the quantum algorithms between the MaxCut and TA problems remains an open question. It might be attributed to the structure of their respective QUBO matrices (see Appendix A.2): they vary in terms of sparsity, connectivity, and the distribution of weights, which can affect the performance of quantum algorithms [64]. This means that these quantum algorithms may need further refinement or a different approach to be effective on such constraint-based problems. Developments in this direction could lead to more effective quantum algorithms for such industrially relevant problems, offering a near-term application of quantum computing in optimization.

The immediate next step in this research would be to test the quantum algorithms for MaxCut instances on real quantum hardware, incorporating adequate error mitigation techniques to account for the noise and errors. While the results from Reference [19] suggest that the normalized Hamiltonian LR-QAOA algorithm can be successfully implemented on real quantum hardware, similar results for the extrapolation approach would be interesting to explore. Additionally, the performance of both algorithms on larger problem sizes, beyond the current limits of quantum simulators, would be a key area of interest. A rigorous benchmarking of the quantum algorithms on real quantum hardware would provide valuable insights into their practical applicability and performance in real-world scenarios, additionally allowing for a more direct comparison with other quantum optimization algorithms [10]. Building on the potential of the LR-QAOA algorithm for solving the MaxCut problem, we cannot discuss any possible quantum advantage [65] over classical algorithms until we have benchmarked these methods.

Furthermore, more advanced sub-sampling techniques could be explored to improve the parameter prediction methods for the LR-QAOA algorithm, and would additionally allow for more types of problems to be solved. For the MaxCut problem, this would enable the algorithm to be tested on graphs with changing densities and structures, to see if the density of the graph has an impact on the performance of the quantum algorithms on MaxCut instances. This in turn would help in understanding the limitations of the LR-QAOA algorithm, and how it can be improved to work better on such problems.

References

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. 27. print. A Series of Books in the Mathematical Sciences. New York [u.a]: Freeman, 2009. 338 pp.

[2] William W. Hager, ed. *Multiscale Optimization Methods and Applications*. Nonconvex Optimization and Its Applications v. 82. New York: Springer, 2006. 407 pp.

[3] Long Le Ngoc Bao, Duc Hanh Le, and Duy Anh Nguyen. "Application of Combinatorial Optimization in Logistics". In: *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*. 2018 4th International Conference on Green Technology and Sustainable Development (GTSD). Nov. 2018, pp. 329–334.

[4] H. P. Williams. *Model Building in Mathematical Programming*. 5th ed. Hoboken, N.J.: Wiley, Jan. 1, 2013. 433 pp.

[5] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, Jan. 1, 1972, pp. 85–103.

[6] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998.

[7] Baha Alzalg. *Combinatorial and Algorithmic Mathematics: From Foundation to Optimization*. 1st ed. Wiley, Oct. 21, 2024.

[8] IBM Corporation. *IBM ILOG CPLEX Optimization Studio*. 12.10. manual. Armonk, NY, USA: IBM Corporation, 2021.

[9] Iain Dunning et al. *MQLib: Implementations of heuristics for the Max-Cut and QUBO problems in C++*. `https://github.com/MQLib/MQLib`. Accessed: 2025-06-27. 2018.

[10] Thorsten Koch et al. *Quantum Optimization Benchmark Library – The Intractable Decathlon*. Apr. 4, 2025. arXiv: `2504.03832 [quant-ph]`. URL: `http://arxiv.org/abs/2504.03832` (visited on 04/24/2025). Pre-published.

[11] Amira Abbas et al. "Challenges and Opportunities in Quantum Optimization". In: *Nature Reviews Physics* 6.12 (Dec. 2024), pp. 718–735.

[12] John Preskill. "Quantum Computing in the NISQ Era and Beyond". In: *Quantum* 2 (Aug. 6, 2018), p. 79.

[13] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. Nov. 14, 2014. arXiv: 1411.4028 [quant-ph]. URL: http://arxiv.org/abs/1411.4028 (visited on 04/23/2025). Pre-published.

[14] Kostas Blekos et al. "A Review on Quantum Approximate Optimization Algorithm and Its Variants". In: *Physics Reports*. A Review on Quantum Approximate Optimization Algorithm and Its Variants 1068 (June 2, 2024), pp. 1–66.

[15] Andrew Lucas. "Ising Formulations of Many NP Problems". In: *Frontiers in Physics* 2 (Feb. 12, 2014).

[16] Fred Glover, Gary Kochenberger, and Yu Du. *A Tutorial on Formulating and Using QUBO Models*. Nov. 4, 2019. arXiv: 1811.11538 [cs]. URL: http://arxiv.org/abs/1811.11538 (visited on 04/24/2025). Pre-published.

[17] Lennart Bittel and Martin Kliesch. "Training Variational Quantum Algorithms Is NP-Hard". In: *Physical review letters* 127.12 (Jan. 1, 2021), p. 120502. PMID: 34597099.

[18] Vladimir Kremenetski et al. *Quantum Alternating Operator Ansatz (QAOA) beyond Low Depth with Gradually Changing Unitaries*. July 22, 2023. arXiv: 2305.04455 [quant-ph]. URL: http://arxiv.org/abs/2305.04455 (visited on 04/23/2025). Pre-published.

[19] J. A. Montanez-Barrera and Kristel Michielsen. *Towards a Universal QAOA Protocol: Evidence of a Scaling Advantage in Solving Some Combinatorial Optimization Problems*. May 15, 2024.

[20] Vanessa Dehn et al. *Extrapolation Method to Optimize Linear-Ramp QAOA Parameters: Evaluation of QAOA Runtime Scaling*. Apr. 11, 2025. arXiv: 2504.08577 [quant-ph]. URL: http://arxiv.org/abs/2504.08577 (visited on 04/24/2025). Pre-published.

[21] Michel X. Goemans and David P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *Journal of the ACM* 42.6 (Jan. 1, 1995), pp. 1115–1145.

[22] Samuel Burer, Renato D. C. Monteiro, and Yin Zhang. "Rank-Two Relaxation Heuristics for MAX-CUT and Other Binary Quadratic Programs". In: *SIAM Journal on Optimization* 12.2 (Jan. 1, 2002), pp. 503–521.

[23] *EnBW GmBH*. URL: https://www.enbw.com/unternehmen/ (visited on 06/18/2025).

[24] Ali Javadi-Abhari et al. *Quantum Computing with Qiskit*. 2024. arXiv: 2405.08810 [quant-ph].

[25] Gerard Sierksma, Gerard Sierksma, and Yori Zwols. *Linear and Integer Optimization: Theory and Practice, Third Edition*. 0th ed. Chapman and Hall/CRC, May 1, 2015.

[26] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. 4th ed. Vol. 21. Algorithms and Combinatorics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[27] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2024.

[28] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10th anniversary ed. Cambridge ; New York: Cambridge University Press, 2010. 676 pp.

[29] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*. 1st ed. Vol. 43. New York, NY, USA: Association for Computing Machinery, May 23, 2023, pp. 143–152.

[30] Thomas H Cormen et al. *Introduction to Algorithms*. MIT press, 2022.

[31] Richard M Karp. "Reducibility among Combinatorial Problems". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer, 2009, pp. 219–241.

[32] Gary Kochenberger et al. "The Unconstrained Binary Quadratic Programming Problem: A Survey". In: *Journal of Combinatorial Optimization* 28.1 (July 1, 2014), pp. 58–81.

[33] Nora Bauer et al. *Solving Power Grid Optimization Problems with Rydberg Atoms*. Apr. 17, 2024. arXiv: `2404.11440 [quant-ph]`. URL: `http://arxiv.org/abs/2404.11440` (visited on 07/02/2025). Pre-published.

[34] Giuseppe Colucci, Stan Van Der Linde, and Frank Phillipson. "Power Network Optimization: A Quantum Approach". In: *IEEE Access* 11 (2023), pp. 98926–98938.

[35] Francisco Barahona et al. "An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design". In: *Operations Research* 36.3 (Jan. 1, 1988), pp. 493–513.

[36] Reinhard Diestel and Reinhard Diestel. *Graph Theory*. 2. ed. Graduate Texts in Mathematics 173. New York, NY: Springer, 2000. 312 pp.

[37] P. Erdős and A. Rényi. "On Random Graphs. I." In: *Publicationes Mathematicae Debrecen* 6.3–4 (July 1, 2022), pp. 290–297.

[38] Iain Dunning, Swati Gupta, and John Silberholz. "What Works Best When? A Systematic Evaluation of Heuristics for Max-Cut and QUBO". In: *INFORMS Journal on Computing* 30.3 (Jan. 1, 2018), pp. 608–624.

[39] Daniel Rehfeldt, Thorsten Koch, and Yuji Shinano. "Faster Exact Solution of Sparse MaxCut and QUBO Problems". In: *Mathematical Programming Computation* 15.3 (Jan. 1, 2023), pp. 445–470.

[40] Paola Festa et al. "Randomized Heuristics for the Max-Cut Problem". In: *Optimization Methods and Software* 17.6 (Jan. 1, 2002), pp. 1033–1058.

[41] Camille Grange, Michael Poss, and Eric Bourreau. "An Introduction to Variational Quantum Algorithms for Combinatorial Optimization Problems". In: *4OR* 21.3 (Jan. 1, 2023), pp. 363–403.

[42] *IBM® Decision Optimization CPLEX® Modeling for Python — IBM® Decision Optimization CPLEX® Modeling for Python (DOcplex) V2.25 Documentation.* URL: http://ibmdecisionoptimization.github.io/docplex-doc/#getting-started-with-docplex (visited on 06/26/2025).

[43] *Tutorial: Linear Programming, (CPLEX Part 1).* URL: https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html (visited on 06/26/2025).

[44] Divya Padmanabhan. "0.878-Approximation for the Max-Cut Problem". Lecture Notes.

[45] Robert M. Freund. *Semidefinite Programming (SDP) and the Goemans-Williamson MAXCUT Paper.* https://ocw.mit.edu/courses/15-099-readings-in-optimization-fall-2003/a837b244b710301b020bab0dd9a92a71_ses1_goemans1.pdf. Lecture, 15.099 Readings in Optimization, Fall 2003. Massachusetts Institute of Technology: MIT OpenCourseWare. License: Creative Commons BY-NC-SA. 2003.

[46] Robert M. Freund. *Semidefinite Optimization.* https://ocw.mit.edu/courses/15-084j-nonlinear-programming-spring-2004/a632b565602fd2eb3be574c537eea095_lec23_semidef_opt.pdf. Lecture, 15.084J Nonlinear Programming, Spring 2004. Massachusetts Institute of Technology: MIT OpenCourseWare. License: Creative Commons BY-NC-SA. 2004.

[47] Nicholas J. Higham. "Analysis of the Cholesky Decomposition of a Semi-definite Matrix". In: ed. by M. G. Cox and S. J. Hammarling. Oxford, UK: Oxford University Press, 1990, pp. 161–185.

[48] Thomas G. Wong. *Introduction to Classical and Quantum Computing.* Omaha, Nebraska: Rooted Grove, 2022. 1 p.

[49] Edward Farhi et al. *Quantum Computation by Adiabatic Evolution.* Jan. 28, 2000. arXiv: quant-ph/0001106. URL: http://arxiv.org/abs/quant-ph/0001106 (visited on 07/07/2025). Pre-published.

[50] Sebastian Brandhofer et al. "Benchmarking the Performance of Portfolio Optimization with QAOA". In: *Quantum Information Processing* 22.1 (Dec. 16, 2022), p. 25. arXiv: 2207.10555 [quant-ph].

[51] Giacomo Nannicini. "Performance of Hybrid Quantum-Classical Variational Heuristics for Combinatorial Optimization". In: *Physical Review E* 99.1 (Jan. 14, 2019), p. 013304.

[52] *Variational Quantum Algorithms | Nature Reviews Physics.* URL: https://www.nature.com/articles/s42254-021-00348-9 (visited on 07/11/2025).

[53] David J Griffiths and Darrell F Schroeter. *Introduction to quantum mechanics.* Cambridge university press, 2018.

[54] Jarrod R McClean et al. "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18.2 (2016), p. 023023.

[55] Jaeho Choi and Joongheon Kim. "A Tutorial on Quantum Approximate Optimization Algorithm (QAOA): Fundamentals and Applications". In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. 2019 International Conference on Information and Communication Technology Convergence (ICTC). Oct. 2019, pp. 138–142.

[56] Leo Zhou et al. "Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices". In: *Physical Review X* 10.2 (Jan. 1, 2020).

[57] Charles R. Harris et al. "Array Programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[58] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17.3 (Mar. 2020), pp. 261–272.

[59] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function Using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[60] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[61] Akshay Agrawal et al. "A Rewriting System for Convex Optimization Problems". In: *Journal of Control and Decision* 5.1 (Jan. 2, 2018), pp. 42–60.

[62] OpenStreetMap Wiki. *Overpass API — OpenStreetMap Wiki*. [Online; accessed 19-July-2025]. 2025.

[63] William Marshall Smart and Robin Michael Green. *Textbook on spherical astronomy*. Cambridge University Press, 1977.

[64] Daniel Müssig et al. *Connecting the Hamiltonian Structure to the QAOA Performance and Energy Landscape*. July 5, 2024. arXiv: `2407.04435 [quant-ph]`. URL: `http://arxiv.org/abs/2407.04435` (visited on 07/28/2025). Pre-published.

[65] Olivia Lanes et al. *A Framework for Quantum Advantage*. July 14, 2025. arXiv: `2506.20658 [quant-ph]`. URL: `http://arxiv.org/abs/2506.20658` (visited on 07/29/2025). Pre-published.

List of Figures

List of Tables

List of Algorithms

Appendix A

Examples of QUBOs

A.1 Examples of QUBO Problem Formulation

A.1.1 Unconstrained objective function

Suppose we have an objective function of the form:

$$F(\mathbf{x}) = x_1^2 + x_2^2 + 3x_1x_2 - 7x_1x_3 + 2x_2x_3 - 9x_3^2$$

where $x_i \in \{0, 1\}$ for $i = 1, 2, 3$. The coefficients of this quadratic polynomial can be written in matrix form as:

$$Q = \begin{pmatrix} 1 & 3 & -7 \\ 0 & 1 & 2 \\ 0 & 0 & -9 \end{pmatrix}$$

This allows us to express the objective function as:

$$\begin{aligned} F(\mathbf{x}) &= x_1^2 + x_2^2 + 3x_1x_2 - 7x_1x_3 + 2x_2x_3 - 9x_3^2 \\ &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 1 & 3 & -7 \\ 0 & 1 & 2 \\ 0 & 0 & -9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ &= \mathbf{x}^T Q \mathbf{x} \end{aligned}$$

If the goal is to minimize F, this optimization problem can thereby be written as:

$$\begin{aligned} \text{minimize} \quad & F(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x} \\ & x_i \in \{0, 1\}, \; i \in \{1, 2, 3\} \end{aligned}$$

Now suppose we have another objective function of the form:

$$F(\mathbf{x}) = 2x_1 + 3x_2 - 4x_3 + 5x_1x_2 - 6x_2x_3 + 22$$

Although this is still a quadratic polynomial, it differs from the previous one in that it includes a constant term and linear terms.

First, note that constant terms affect all costs equally, and thus does not affect the optimization problem. If it is important to get the optimal cost value, one can simply add the constant term to the objective function after finding the optimal solution. Secondly, with the fact that the square of a binary variable is equal to the variable itself, we can rewrite the objective function as:

$$\begin{aligned}
F(\mathbf{x}) &= 2x_1 + 3x_2 - 4x_3 + 5x_1x_2 - 6x_2x_3 + 22 \\
&= 2x_1^2 + 3x_2^2 - 4x_3^2 + 5x_1x_2 - 6x_2x_3 + 22 \\
&= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 2 & 5 & 0 \\ 0 & 3 & -6 \\ 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + 22 \\
&= \mathbf{x}^T Q \mathbf{x} + 22
\end{aligned}$$

Summarizing, we can write the QUBO problem as:

$$\begin{aligned}
\text{minimize} \quad & F(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x} + c \\
& x_i \in \{0, 1\},\ i \in \{1, 2, 3\}
\end{aligned}$$

Therefore we see that the QUBO framework is a useful way to represent COPs with quadratic objective functions with binary variables. However, a great advantage of the QUBO framework is that it can even be used to represent problems that have constraints on the variables. This is achieved by transforming the constraints into a penalty term that is added to the objective function. In the case of a minimization problem, this penalty term is designed such that it increases the cost of the objective function whenever a constraint is violated, thereby discouraging solutions that do not satisfy the constraints.

A.1.2 QUBO with constraints

Consider the following example of a quadratic binary optimization problem in N binary variables with a constraint:

$$\begin{aligned}
\text{minimize} \quad & \tilde{F}(\mathbf{x}) \\
\text{subject to} \quad & x_1 + x_2 \leq 1 \\
& x_1, x_2 \in \{0, 1\},\ i \in \{1, \cdots, N\}
\end{aligned}$$

To transform this problem into a QUBO problem, we can introduce a penalty term to the objective function of the form:

$$P(\mathbf{x}) = \Lambda \cdot (x_1 x_2)$$

where Λ is a large positive constant that determines the strength of the penalty. The new objective function can then be written as:

$$\begin{aligned} F(\mathbf{x}) &= \tilde{F}(\mathbf{x}) + P(\mathbf{x}) \\ &= \tilde{F}(\mathbf{x}) + \Lambda \cdot (x_1 x_2) \\ &= \mathbf{x}^T Q \mathbf{x} + \Lambda \cdot (x_1 x_2) \\ &= \mathbf{x}^T Q' \mathbf{x} \end{aligned} \tag{A.1}$$

This transformation allows us to express the problem in the QUBO framework, where Q' is a new matrix that incorporates the penalty term. The factor Λ should be chosen large enough to ensure that the penalty term dominates the objective function when the constraint is violated. The choice of Λ is dictated by the specific problem and the scale of the objective function, and it is often determined through experimentation or domain knowledge. The penalty factor formulation for many famous COPs is well-studied as can be seen in [16].

A.2 Examples of QUBOs in this work

The structure of the QUBOs used in this thesis differ between each problem. Here we give examples of how the QUBO for the weighted complete MaxCut, weighted R3R MaxCut, and the TA problem look like in terms of general structure and density of QUBO weights.

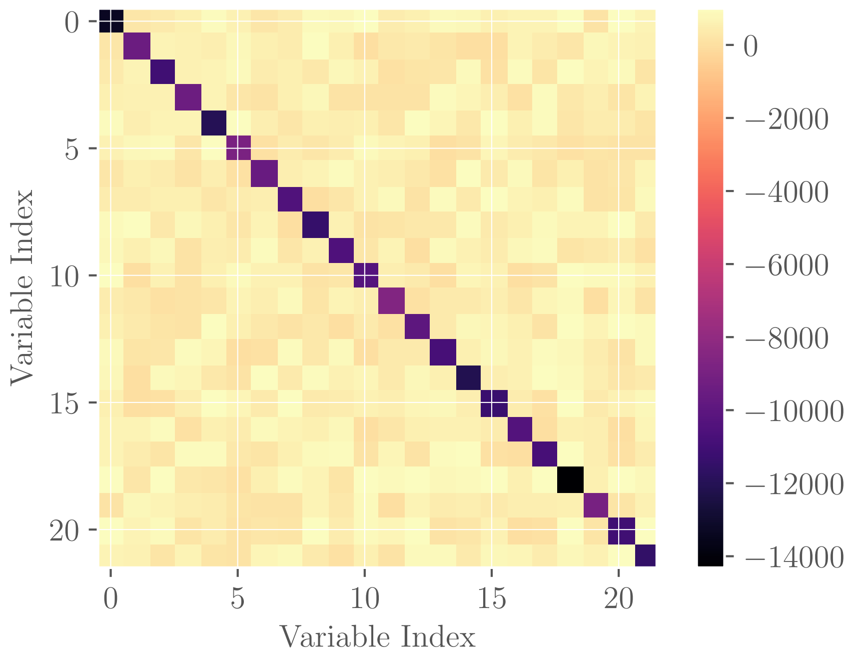


Figure A.1: The QUBO matrix for the weighted complete Max-Cut problem with 22 nodes, plotted as a heatmap.

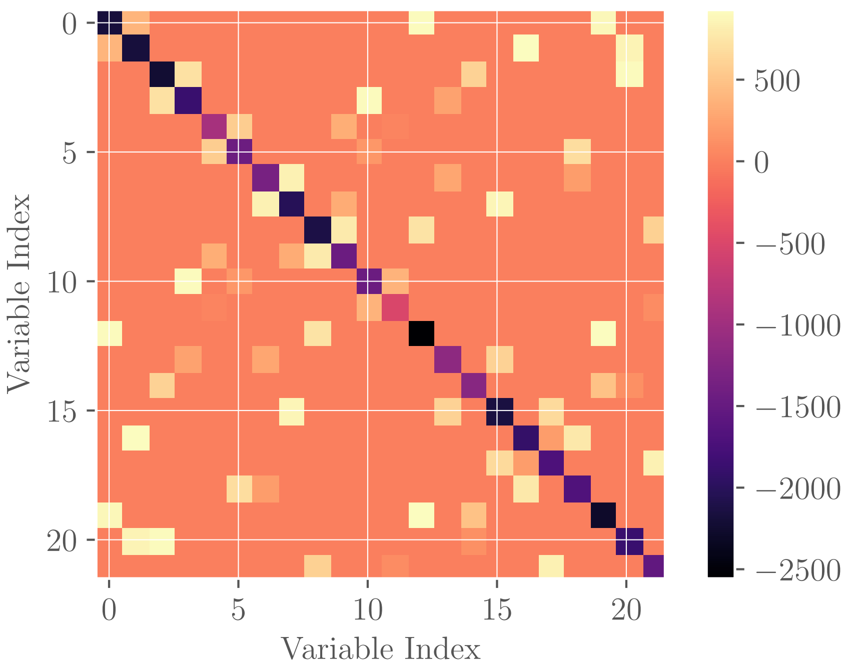


Figure A.2: The QUBO matrix for the weighted R3R MaxCut problem with 22 nodes, plotted as a heatmap.

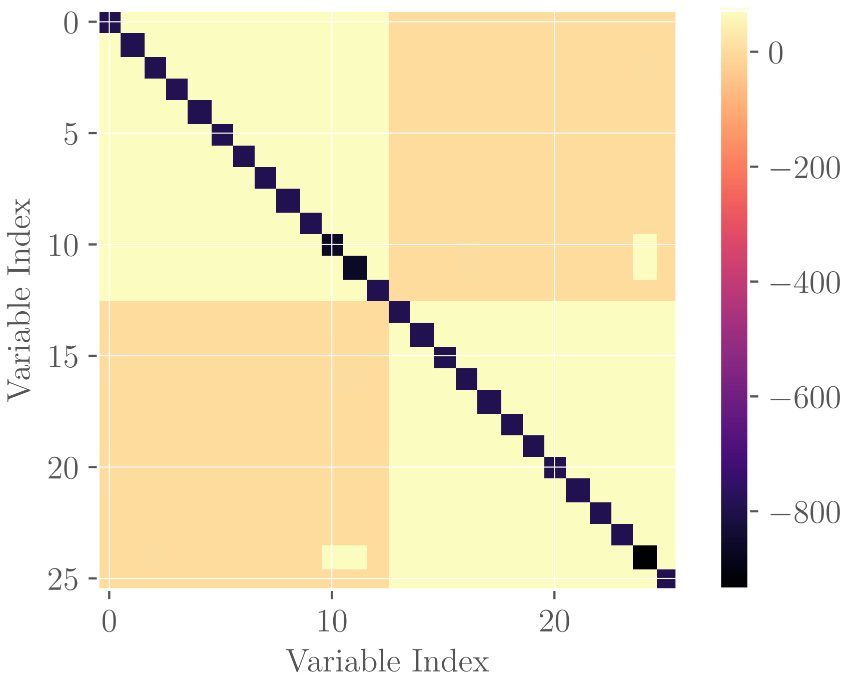


Figure A.3: The QUBO matrix for the Technician-Asset allocation problem with 13 technicians and 13 assets, (with constraints T_1 and A_1 set to 10 each) plotted as a heatmap. The matrix structure varies from the previous two examples, and the structure could be further analyzed to determine its significance in the quantum optimization process.

Appendix B

CPLEX Example Log

An example of a CPLEX log during the solution of a weighted complete MaxCut QUBO problem is shown below. The solver (which looks for optimal solutions until the gap becomes the specified tolerance 10^{-6}), did not find an optimal solution, even after 47,650 seconds (or 13.2 hours) of computation time, at which point, the gap was still 0.91%. The output is truncated to show only the first 80 lines and then the last line when the solver was stopped.

```
Running CPLEX on QUBO instance 0 of size 100.

Version identifier: 22.1.0.0 | 2022-03-27 | 54982fbec
CPXPARAM_Read_DataCheck                                 1
Found incumbent of value 0.000000 after 0.00 sec. (0.04 ticks)
Found incumbent of value -0.000000 after 0.00 sec. (0.08 ticks)
Tried aggregator 1 time.
MIP Presolve added 9900 rows and 4950 columns.
Reduced MIP has 9900 rows, 5050 columns, and 19800 nonzeros.
Reduced MIP has 5050 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (1.35 ticks)
Probing time = 0.01 sec. (0.60 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 4950 rows and 0 columns.
Reduced MIP has 4950 rows, 5050 columns, and 14850 nonzeros.
Reduced MIP has 5050 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.01 sec. (8.89 ticks)
Classifier predicts products in MIQP should not be linearized.
Represolve...

Tried aggregator 1 time.
Repairing indefinite Q in the objective.
Reduced MIQP has 0 rows, 100 columns, and 0 nonzeros.
```

```
Reduced MIQP has 100 binaries, 0 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 10000 nonzeros.
Presolve time = 0.00 sec. (8.36 ticks)
Probing time = 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
Reduced MIQP has 0 rows, 100 columns, and 0 nonzeros.
Reduced MIQP has 100 binaries, 0 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 10000 nonzeros.
Presolve time = 0.00 sec. (4.74 ticks)
Probing time = 0.00 sec. (0.01 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 32 threads.
Root relaxation solution time = 0.03 sec. (5.64 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                          0.0000     -4953.7665           ----
      0     0    -1385.5045   100      0.0000     -1385.5045        7    ----
*     0+    0                      -1323.9565     -1385.5045            4.65%
*     0+    0                      -1324.6092     -1385.5045            4.60%
*     0+    0                      -1348.1473     -1385.5045            2.77%
*     0+    0                      -1349.2403     -1385.5045            2.69%
      0     2    -1385.5045   100  -1349.2403     -1385.5045        7     2.69%
Elapsed time = 14.37 sec. (99.45 ticks, tree = 0.02 MB, solutions = 5)
*    55+    9                      -1349.8612     -1385.0422            2.61%
*   158+   27                      -1349.8974     -1384.8067            2.59%
*   526+  108                      -1351.3274     -1384.8067            2.48%
```

```
*  1289+   690                             -1351.4880     -1384.8067          2.47%
   4194   2031     -1359.6661     63       -1351.4880     -1382.6273      6770  2.30%

Performing restart 1

Repeating presolve.
Tried aggregator 1 time.
Reduced MIQP has 0 rows, 100 columns, and 0 nonzeros.
Reduced MIQP has 100 binaries, 0 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 10000 nonzeros.
Presolve time = 0.00 sec. (4.74 ticks)
Tried aggregator 1 time.
Reduced MIQP has 0 rows, 100 columns, and 0 nonzeros.
Reduced MIQP has 100 binaries, 0 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 10000 nonzeros.
Presolve time = 0.00 sec. (4.74 ticks)
Represolve time = 0.01 sec. (11.00 ticks)
   7440    243     -1364.6926     61       -1351.4880     -1381.7755     21648  2.24%
*  9556+  1436                             -1351.5478     -1381.7755          2.24%
  16078   7236     -1373.1297     78       -1351.5478     -1381.0907     47257  2.19%
  31550  19601     -1374.1379     85       -1351.5478     -1377.9524     91067  1.95%
* 36840+23495                              -1351.8274     -1377.7113          1.91%
  42432  26823     -1356.3093     45       -1351.8274     -1377.4994    117689  1.90%
  54746  38840     -1363.5332     60       -1351.8274     -1377.0841    160732  1.87%
  66020  50517     -1373.8947     80       -1351.8274     -1376.7820    202124  1.85%
  73597  56122     -1374.1253     81       -1351.8274     -1376.7820    222059  1.85%
  81867  63251     -1355.0936     41       -1351.8274     -1376.7820    247950  1.85%
```

```
 113522  91741     -1361.8101     64     -1351.8274     -1376.7820    350238    1.85%
...
...
...
...
...
Elapsed time = 47646.99 sec. (4604954.09 ticks, tree = 46545.55 MB, solutions
Nodefile size = 44497.24 MB (26933.24 MB after compression)
 377797372 300313212     -1359.6892     57     -1351.8274     -1364.1135  9.24e+08    0.91%
 379199034 301392974     -1357.6720     57     -1351.8274     -1364.1080  9.27e+08    0.91%
 380437929 302387953     -1363.0123     62     -1351.8274     -1364.1025  9.30e+08    0.91%
```

Appendix C

Scaling of depths with increasing p

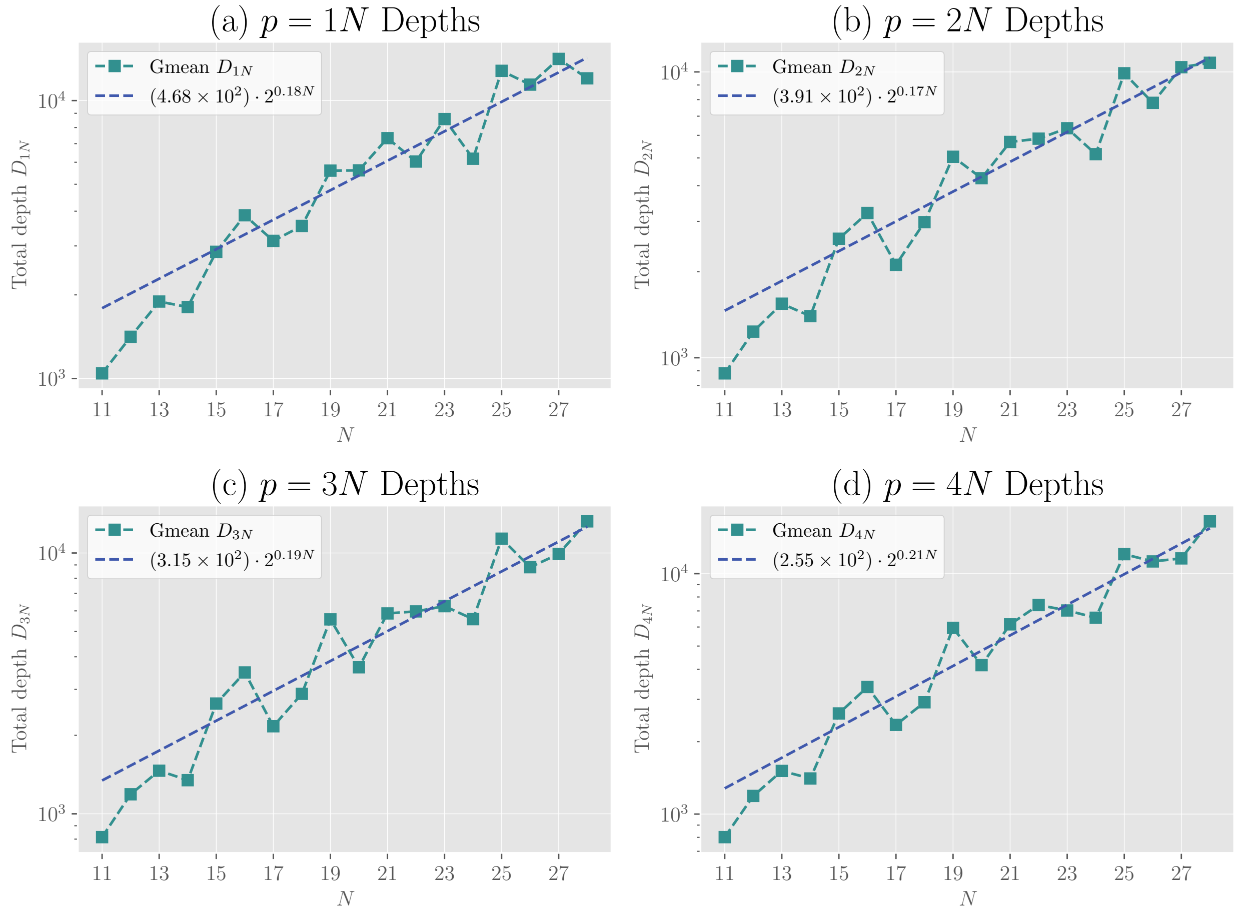


Figure C.1: Scaling of depths with increasing p for Normalized Hamiltonian LR-QAOA, weighted complete MaxCut problem. The depths are shown for $p = 1N$, $2N$, $3N$ and $4N$. The fitted exponential and polynomial curves are shown for each depth variant.