CHALMERS
UNIVERSITY OF TECHNOLOGY

KU LEUVEN
FACULTY OF ENGINEERING SCIENCE

Parameter Setting Strategies in Multi-Angle Quantum Approximate Optimization Algorithm

Yash Vardhan Bhobia

Thesis submitted as part of the degree Master of Science: Erasmus Mundus Master of science in Nanoscience and Nanotechnology, with elective Quantum Computing

Supervisor:
Dr. Laura Gárcia Alvarez

Referee:
Prof. Thilo Bauch

KUL Promotor:
Prof. Roel Van Beeumen

PhD Advisor:
Isak Lyngfelt

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Erasmus Mundus

Published by Yash Vardhan Bhobia,
Faculty of Engineering Science, Kasteelpark Arenberg 10 bus 2440, B-3001 Leuven

Preface

This thesis was written as part of the Erasmus Mundus Master of Science in Nanoscience and Nanotechnology, hosted jointly by Chalmers University of Technology and KU Leuven. The research was carried out at the Department of Microtechnology and Nanoscience at Chalmers, under the daily supervision of Isak Lyngfelt, with Prof. Roel Van Beeumen as supervisor at KU Leuven.

I am grateful to Dr. Laura García Álvarez for many enlightening discussions and for her guidance and support throughout this work, and to Isak Lyngfelt for his patience, detailed feedback, and the many hours spent discussing both the ideas and the code behind them.

Finally, I thank my family for their patience and encouragement across the distance, and my friends for the laughter, perspective, and late-night conversations that kept me grounded.

Yash Vardhan Bhobia

Contents

Abstract

Combinatorial optimization problems such as Max-Cut are difficult to solve exactly on classical computers, and variational quantum algorithms like Quantum Approximate Optimization Algorithm (QAOA) offer a promising avenue to solve these problems on near-term quantum computers. The QAOA prepares a parameterized quantum state by alternating two sets of unitaries, one encoding the problem cost and one mixing between candidate solutions, while a classical optimizer tunes the variational parameters associated with the unitaries to minimize a cost function. Standard QAOA shares one parameter per layer across each unitary, using only $2p$ parameters for a circuit of depth p. Its generalization, multi-angle QAOA (ma-QAOA), assigns an independent parameter to every quantum gate in every layer, increasing the number of variational parameters being optimized by the classical optimizer. This added freedom improves solution quality, but increases the dimensionality of the classical optimization problem. Moreover, the larger parameter space increases susceptibility to barren plateaus, where cost-function gradients vanish exponentially and impede convergence. As such, it is important to find good parameter setting strategies to leverage the added expressibility of ma-QAOA, while minimizing the added expensive classical computation.

This thesis studies several strategies for setting ma-QAOA parameters efficiently for the Max-Cut problem on random 3-regular graphs. The first strategy builds on the fact that the QAOA cost function can be written as a sum of local terms. For Max-Cut, this means the total cost is a sum over local subgraphs whose size depends on the circuit depth p. We pre-optimize parameters on a library of all unique local subgraphs, then for each edge of a target host graph we look up the corresponding entry in the library and transfer those parameters directly into the full ma-QAOA circuit skipping the classical optimization step. We also consider variants of this idea, such as transferring only the cost-unitary parameters while optimizing the mixer-unitary parameters, or the reverse, and using the transferred angles as warm-starts for full classical optimization. Since the library size grows exponentially with p, we also study extrapolation of library parameters from depth $p = 2$ to $p = 3$. The second strategy uses a linear-ramp schedule, where each edge and node parameter increases linearly across layers, so that the parameter count becomes independent of depth. All methods are compared against fully optimized QAOA, fully optimized ma-QAOA, and fixed-angle QAOA, where a single pair of angles is shared across all gates and layers. The comparison is carried out in exact statevector simulations on graphs with up to 20 vertices, using approximation ratio and classical runtime as metrics. We find that none of these strategies recover the performance advantage of fully optimized ma-QAOA. Across all methods, the approximation ratio stays close to that of standard QAOA and previously studied QAOA transfer schemes, rather than approaching ma-QAOA. The transferred parameters do, however, serve well as a warm-start initialization for full optimization.

Keywords: Noisy Intermediate Scalable Quantum, Quantum Approximate Optimization Algorithm, Quantum Computing, Max-Cut problem, Barren Plateau, Dynamical Lie Algebra.

List of Figures

[1] `ma_qaoa_opt` at $p = 3$ is omitted from this plot because it was run on separate hardware due to very long simulation time requirements, and hence can not be compared on the same scale.

Chapter 1

Introduction

1.1 Quantum Computing and Combinatorial Optimization

Quantum computing is expected to bring unprecedented computational capabilities to applications across multiple scientific and industrial domains. In chemistry and materials science, quantum simulations of molecular reactions promise advances in drug discovery (7–10) and the design of novel materials with tailored functionalities (11–13). In cryptography, Shor's polynomial-time factoring algorithm (14) can break RSA-based protocols in practical time frames, a task that remains intractable for classical brute-force methods. In artificial intelligence and quantum machine learning, quantum processors offer novel approaches to regression, classification, and clustering (15–17). Furthermore, in engineering and operations research, they demonstrate significant potential for optimization, scheduling, and resource allocation (18–21).

This thesis focuses on combinatorial optimization, a class of problems with substantial economic and logistical relevance. Many real-world tasks, such as vehicle routing, portfolio selection, network design, and graph partitioning, can be formulated as optimizing a cost function over an exponentially large solution space. Most such problems are classified as NP-hard (see Section 2.1), meaning that finding an exact solution using classical algorithms scales exponentially in time with the input size. Throughout this thesis, we work with the Maximum Cut (Max-Cut) problem on 3-regular graphs as our primary benchmark. Max-Cut is selected because it is structurally simple, NP-hard, admits a natural Ising formulation (22), and is one of the most extensively studied NP-hard problems in the context of quantum algorithms. Its connection to physics makes Max-Cut highly suitable for quantum hardware. Specifically, its cost function maps directly to the Ising model, which describes the energy of a system composed of interacting two-state spins, typically denoted as "spin up" or "spin down" and often subject to an external field. The total energy comprises two components: the local energy of individual spins and the interaction energy between spin pairs.

All possible configurations of the system can be represented as combinations of these spin states. Consequently, these two states form the computational basis for all possible system configurations. The pairwise interaction energy is diagonal in this computational basis, meaning it assigns an energy value to a given configuration without causing transitions between different states. The specific Max-Cut problem we consider features a cost function analogous to the Ising model with only two-body interactions. This structural similarity makes it highly suitable for encoding on quantum hardware. In quantum optimization, the central idea is to encode this cost function into a quantum operator called

the Cost Hamiltonian, denoted $\langle H_C \rangle$. Preparing the ground state of $\langle H_C \rangle$ is then mathematically equivalent to minimizing the original optimization problem (22).

The challenge then becomes how to efficiently prepare, or closely approximate, this ground state on near-term quantum devices. The remainder of this chapter motivates the specific algorithm studied in this thesis. It does so by reviewing the constraints of near-term quantum hardware, the variational paradigm designed to mitigate these limitations, and the parameter-scaling problem that motivates our parameter setting strategies.

1.2 The NISQ Era and Variational Quantum Algorithms

Current quantum hardware remains too noisy to run large-scale algorithms reliably. This defines the Noisy Intermediate-Scale Quantum (NISQ) era: devices with limited qubit counts, short coherence times, and imperfect gate fidelities, insufficient for fully fault-tolerant computation (23). For instance, state-of-the-art superconducting devices have only recently surpassed one thousand physical qubits, with stable operation lasting only a few milliseconds (24). Since gate operations take anywhere from tens of nanoseconds to several microseconds depending on the platform, the effective circuit depth before decoherence destroys the quantum state is limited to at most a few hundred gates. To overcome these limitations, significant effort is directed toward fault-tolerant quantum computation, where logical qubits are encoded across multiple noisy physical qubits to enable error correction. However, achieving full fault tolerance remains a long-term goal.

To utilize this imperfect near-term hardware, a hybrid quantum-classical paradigm has emerged, giving rise to the family of Variational Quantum Algorithms (VQAs) (25). This architecture combines the strengths of both quantum and classical processors in a closed loop: the Quantum Processing Unit (QPU) prepares and measures a parameterized quantum state, and a classical optimizer updates the circuit's gate parameters to minimize the expectation value of a problem-specific cost Hamiltonian $\langle H_C \rangle$. By offloading the optimization to a classical machine, VQAs can operate with shallower circuits, keeping quantum resource requirements within NISQ limits. A prominent example is the Quantum Approximate Optimization Algorithm (QAOA) (26), the canonical VQA for combinatorial optimization. QAOA prepares a parameterized state by alternating p layers of cost evolution under $\langle H_C \rangle$ and with p layers of evolution under a mixing Hamiltonian $\langle H_M \rangle$. Standard QAOA applies a uniform rotation across all gates of a given type within a layer, requiring only $2p$ scalar parameters to be optimized classically.

Despite their potential, VQAs face severe trainability challenges. The barren plateau phenomenon causes the cost landscape to become exponentially flat as the system scales, making gradient-based optimizers to stall (27). Additionally, as the number of variational parameters increases, the classical optimizer struggles to navigate the high-dimensional search space, which is often riddled with local minima.

To increase the expressivity of the parametrized state, another variant of the algorithm, multi-angle QAOA (ma-QAOA) was introduced (28). Ma-QAOA lifts the uniform-parameter restriction of standard QAOA and each individual gate is assigned its own independent classical parameter. While this extra freedom allows ma-QAOA to explore more of the optimization landscape and achieve substantially better performance at similar circuit depths (29,30), it causes the parameter space to scale exponentially with the size of the physical system. This parameter explosion increases the risk of barren plateaus (see Appendix B) and makes classical training prohibitively expensive. Furthermore, because variational optimization landscapes are often riddled with local minima, classical training typically requires multiple optimization restarts from different random configurations to find a high-quality minimum. Navigating such a high-dimensional space across multiple restarts quickly

becomes classically prohibitive. Developing parameterised circuits that retain enough expressivity to capture high-quality solutions while maintaining a navigable optimization landscape is one of the central challenges of the NISQ era, and motivates the work in this thesis as well.

1.3 Our Work

To resolve this bottleneck, we introduce two parameter-setting strategies that bypass full whole-system optimization. Both strategies leverage specific structural properties of the QAOA circuit.

For the Max-Cut problem, the graph-to-circuit mapping assigns each node to a single qubit and each edge to a two-qubit gate. Due to the Ising formulation, the cost Hamiltonian decomposes into a sum of local edge operators. By the linearity of expectation, the total expectation value of the cost function is simply the sum of the expectation values of the individual edges. Furthermore, because of the layered structure of the quantum circuit, the expectation value of any single edge depends only on the gates within a distance p in the graph. This *light-cone* property means the optimal parameters for an edge are completely determined by the nodes and edges within its p-depth neighborhood, regardless of the overall graph size.

This observation leads to our first strategy, the **subgraph-transfer technique**. Focusing on 3-regular graphs, we precompute and maintain a library of all possible p-depth subgraphs built around a central edge. We optimize these small subgraphs, extract the optimal parameters, and store them. To evaluate a new, large graph, we extract the p-depth subgraph around each edge, match it to its corresponding library entry, and assign those pre-optimized parameters to the target circuit. By constructing this library once, we completely skip the expensive classical optimization loop for ma-QAOA on any graph of this family.

The second strategy, **slope-constrained optimization**, is based on a different observation of QAOA parameter trends. In standard QAOA, optimal parameters across successive layers often follow a smooth, linear trend, a “linear ramp”, from layer to layer (5). We extend this concept to the multi-angle case by constraining the parameters of each individual edge and node to follow a linear slope across the p layers. Instead of optimizing an independent parameter for every single gate in every layer, we optimize only a single slope for each edge and a single slope for each node. This keeps the classical parameter count tied solely to the number of edges and nodes in the graph, making it independent of the circuit depth p. This constraint dramatically simplifies the optimization landscape and reduces the severity of barren plateaus, while still retaining localized expressivity across the graph.

1.4 Thesis Outline

This thesis is organised as follows. Chapter 2 reviews the required theory: complexity classes, quantum computing basics, the Maxcut problem and its Ising formulation, standard QAOA, multi-angle QAOA, the light-cone locality principle that motivates our first strategy, and the related parameter-setting literature. Chapter 3 describes the simulation setup: graph generation, the subgraph-library construction and transfer pipeline, their semi-transfer variants, extrapolation to larger depths, and warm-start variants, the linear-ramp and slope-constrained optimization strategies, the simulation backends, and the evaluation metrics. Chapter 4 presents the benchmark results, comparing approximation ratios and runtimes of all methods on random 3-regular graphs, and analyses why each strategy behaves as it does. Chapter 5 summarizes the findings, states the limitations of this work, and outlines directions for future research. The appendices collect the formal derivation of the reverse

causal cone bound, the dynamical Lie algebra treatment of barren plateaus, the graph-isomorphism and Weisfeiler-Lehman hashing tools used in library matching, and a preliminary extension of the transfer benchmark to larger graphs using matrix-product-state simulation.

Chapter 2

Background and Related Work

This chapter provides the theoretical foundations and background required to understand the methodologies and results presented in this thesis.

2.1 Complexity Classes

Computational problems are grouped into complexity classes according to how the resources required to solve them scale with the input size.

The class P contains decision problems solvable by a deterministic Turing machine in polynomial time, representing problems that are considered easily solvable in a classical setting. NP (non-deterministic polynomial time) contains decision problems whose candidate solutions can be verified in polynomial time, even though finding those solutions from scratch may require exponential time. A problem is NP-hard if every problem in NP can be reduced to it in polynomial time; such a problem is at least as difficult as the hardest problems in NP, but does not need to be a decision problem or lie in NP itself. The intersection of NP and NP-hard defines NP-complete, the class of decision problems in NP to which every other NP problem can be reduced.

For optimization problems, a relevant complexity class is APX (Approximable), which contains optimization problems that admit constant-factor approximations in polynomial time. A problem is APX-hard if every problem in APX reduces to it; for such problems, there exists a constant threshold beyond which polynomial-time classical approximation is mathematically impossible unless $P =$ NP. Figure 2.1 illustrates the conjectured relationships among these classical complexity classes.

Max-Cut, the problem studied in this thesis, is a classic NP-hard optimization problem (22). The best-known classical polynomial-time algorithm for general Max-Cut is the randomized semidefinite programming algorithm by Goemans and Williamson (31), which guarantees an approximation ratio of approximately 0.878 on general graphs with non-negative edge weights. Having an upper bound on approximation places Max-Cut within the APX-hard class and finding exact solutions in polynomial time (unless $P = NP$), or even approximations arbitarily close to optimal, is classically not possible for large systems.

This places Max-Cut in the regime where the standard Quantum Approximate Optimization Algorithm (QAOA) and its multi-angle generalization (ma-QAOA) are natural candidates. However, it should be noted that the classical optimization step of training these parameterized quantum circuits is itself NP-hard (32). We therefore study cheap, non-trivial parameter-setting heuristics to make near-term quantum optimization more viable.

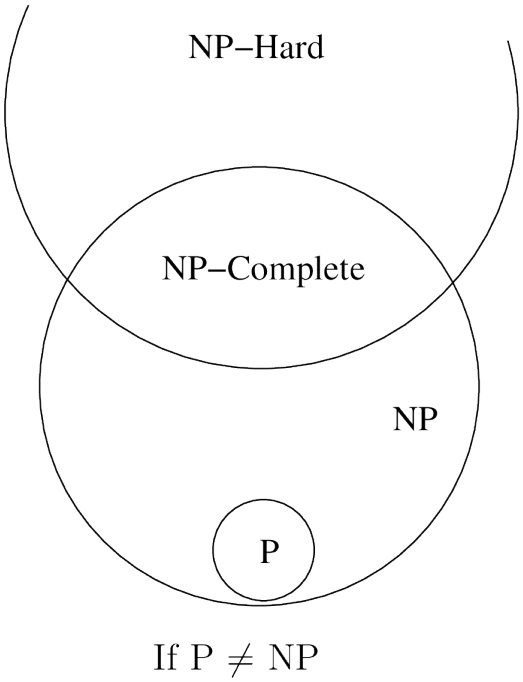

Figure 2.1: Conjectured relationship between different classical and quantum complexity classes. The boundaries are valid under the conjecture that $P \neq \text{NP}$. Image taken from (1)

2.2 Quantum Computing Basics

2.2.1 Quantum bits

Classical computers store information in bits, which take deterministic values of 0 or 1, physically which is usually realized through voltage levels in a circuit. In a quantum computer, information is instead stored in quantum bits, or qubits, denoted as $|0\rangle$ or $|1\rangle$. These qubits are quantum-mechanical two-level systems which can be physically realized using discrete atomic energy levels, Cooper-pair states in superconducting transmon circuits, electron or nuclear spins, nitrogen-vacancy (NV) centers in diamond, or photonic states (33,34).

Measuring a qubit yields one of two outcomes, $|0\rangle$ or $|1\rangle$, which form the computational basis states (33,34). Unlike a classical bit, however,before measurement a qubit can exist in a linear combination, or superposition, of these basis states:

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

where $\alpha, \beta \in \mathbb{C}$ are complex probability amplitudes. To ensure total probability equals one, these amplitudes satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. When the qubit is measured in the computational basis, the outcome $|0\rangle$ occurs with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. The act of measurement itself causes the superposition state to collapse into the observed basis state.

The pure quantum state shown in Equation 1 can also be represented geometrically on a unit sphere known as the Bloch sphere (Figure 2.2). Parameterized by polar angle θ and azimuthal angle ϕ, the superposition state can be written as:

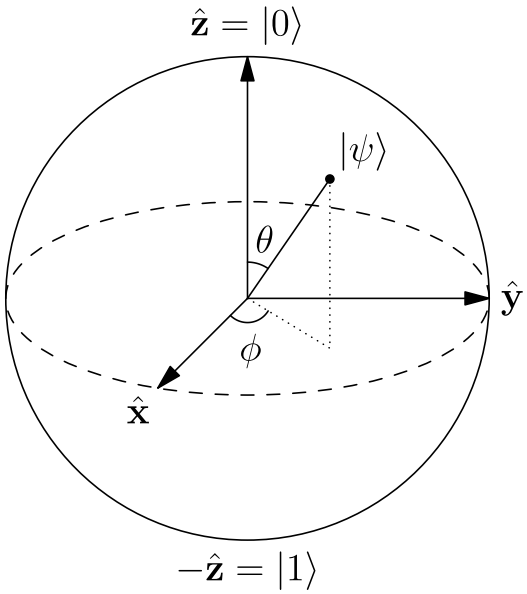

Figure 2.2: Representation of a pure state $|\psi\rangle$ on the Bloch sphere, where θ is the polar angle and ϕ is the azimuthal angle (2).

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

with $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$ (34). This geometric picture makes it intuitive to visualize quantum operators which are the quantum analogues of classical logic gates. We will go in more detail on quantum operators in Section 2.2.3.

2.2.2 Hilbert space and multi-qubit states

A Hilbert space is the complex vector space containing all valid state vectors of a quantum system. The state space of a single qubit is the two-dimensional complex Hilbert space $\mathbb{C}^2$ spanned by the computational basis states $|0\rangle$ and $|1\rangle$. For an n-qubit system, the total state space is formed by taking the n-fold tensor product $\left(\mathbb{C}^2\right)^{\otimes n}$, resulting in a 2^n-dimensional Hilbert space. The tensor product combines the individual qubit spaces such that each basis vector $|x\rangle = |x_1\rangle \otimes |x_2\rangle \otimes ... \otimes |x_n\rangle$ corresponds to a distinct n-bit binary string $x \in \{0,1\}^n$. An arbitrary pure n-qubit state is a normalized vector in this combined space:

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle, \tag{3}$$

where the complex amplitudes α_x satisfy $\sum_x |\alpha_x|^2 = 1$. The summation runs over all 2^n computational basis states.

The exponential growth of the Hilbert space dimension with system size n (or simply, number of qubits) is a primary source of potential quantum speed-up, allowing a quantum state to encode a superposition over exponentially many classical configurations simultaneously. However, this high dimensionality also presents severe challenges. From a computational perspective, simulating these 2^n amplitudes classically becomes intractable for more than a few dozen qubits. From an algorithmic perspective, while the classical optimizer searches over a low-dimensional parameter space, the quantum state evolves inside this vast Hilbert space. As n grows, this can lead to concentration-

of-measure phenomena, leading to barren plateaus and dense local minima that make parameter landscapes increasingly difficult to navigate (34,35).

2.2.3 Quantum gates and circuits

In classical computation, logical operations are performed by logic gates such as AND, OR, NOT, and XOR, which deterministically manipulate binary bits. In a quantum system, state manipulation is performed by quantum gates, which act on complex probability amplitudes. The time evolution of a quantum state is governed by the time-dependent Schrödinger equation (4,33,36):

$$i\hbar \frac{\mathrm{d} \mid \psi(t)\rangle}{\mathrm{d}t} = \hat{H} \mid \psi(t)\rangle, \tag{4}$$

where $\hbar$ is the reduced Planck constant and $\hat{H}$ is the system Hamiltonian, whose eigenvalues represent the allowed energy levels of the quantum system. For an n-qubit system, the Hamiltonian is a $2^n \times 2^n$ Hermitian operator with 2^n real energy eigenvalues satisfying $\hat{H} \mid \psi_i\rangle = E_i \mid \psi_i\rangle$.

When the Hamiltonian $\hat{H}$ is time-independent (and adopting natural units where $\hbar = 1$), integrating Equation 4 yields a unitary time-evolution operator:

$$\hat{U}(t) = e^{-i\hat{H}t}. \tag{5}$$

If we know the initial state $|\psi(0)\rangle$, the state at a later time t is obtained by applying this unitary operator:

$$\mid \psi(t)\rangle = \hat{U}(t) \mid \psi(0)\rangle. \tag{6}$$

Quantum gates are physical implementations of these unitary operators $\hat{U}$ acting on one or more qubits, satisfying the unitarity condition $\hat{U}^\dagger \hat{U} = \hat{U}\hat{U}^\dagger = I$. Because unitaries preserve vector norms, quantum gate operations are inner-product-preserving and reversible, acting by rotating the state vector in Hilbert space without changing its magnitude.

Single-qubit gates are represented by 2×2 unitary matrices. The most fundamental are the Pauli matrices, defined as:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{7}$$

The computational basis states introduced earlier, $|0\rangle = (1, 0)^T$ and $|1\rangle = (0, 1)^T$, are the eigenstates of the Pauli Z matrix with eigenvalues $+1$ and -1, respectively. Another essential single-qubit gate is the Hadamard gate $H = \frac{X+Z}{\sqrt{2}}$, which creates an equal superposition when applied to a basis state:

$$H|0\rangle = |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad \& \quad H|1\rangle = |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{8}$$

Evolving a qubit state under a general Pauli matrix $P \in \{X, Y, Z\}$ for a duration $t = \frac{\theta}{2}$ produces a parameterized rotation gate:

$$R_{P(\theta)} = e^{-i\left(\frac{\theta}{2}\right)P}, \tag{9}$$

which rotates the state vector by an angle θ about the corresponding axis of the Bloch sphere in a counterclockwise direction (34).

Multi-qubit gates couple distinct qubits and generate quantum entanglement. A primary example is the controlled-NOT (CNOT) gate, which flips the target qubit if and only if the control qubit is in the state $|1\rangle$. Together with arbitrary single-qubit rotations, CNOT forms a universal gate set, meaning any n-qubit unitary operation can be compiled into a circuit built from these components to arbitrary precision.

In this work, we apply QAOA to the Max-Cut problem, whose circuit includes a two-qubit ZZ phase rotation. The corresponding unitary is $e^{-i\left(\frac{\gamma}{2}\right)Z_i Z_j}$, acting on qubit pairs (i, j). At the hardware

level, a two-qubit ZZ rotation gate $R_{Z_iZ_j}(\theta) = e^{-i\left(\frac{\theta}{2}\right)Z_iZ_j}$ (it is of same form as in Equation 9) is typically compiled using two CNOT gates and a single-qubit $R_{Z(\theta)}$ rotation on the target qubit as:

$$R_{Z_iZ_j}(\theta) = \text{CNOT}(i \to j) R_{Z_{j(\theta)}} \text{CNOT}(i \to j). \tag{10}$$

A quantum circuit consists of a structured sequence of such single- and multi-qubit gates applied to an initial state, followed by quantum measurement. Measuring an n-qubit state in the computational basis collapses the wave function and yields a classical binary string $x \in \{0,1\}^n$, with probability $P(x) = |\alpha_x|^2 = |\langle x \mid \psi \rangle|^2$.

For variational quantum algorithms, the objective is to sample individual output strings and then to evaluate the expectation value $\langle \hat{O} \rangle$ of a problem-specific observable $\hat{O}$ (such as a cost Hamiltonian H_C):

$$\langle \hat{O} \rangle = \langle \psi \mid \hat{O} \mid \psi \rangle = \sum_{x \in \{0,1\}^n} O(x) P(x). \tag{11}$$

In practice, this expectation value is estimated by repeatedly executing the quantum circuit, taking measurement shots, and averaging the corresponding classical cost values $O(x)$. This empirical expectation value serves as the classical feedback signal passed to the classical optimizer in VQAs like QAOA (34,35).

2.3 Max-Cut

Having established the fundamentals of quantum computing and the computational complexity limits of classical solvers, we now formally define the Maximum Cut (Max-Cut) problem. Mathematically formulating the problem is the necessary first step toward encoding its cost function into a quantum system.

Consider an undirected, unweighted graph $G = (V, E)$, where V represents the set of vertices (or nodes) and E represents the set of edges connecting those vertices. A cut is defined as a partition of the vertex set V into two disjoint subsets, S and $V \setminus S$ (the vertices not in S). The cut value $C(S)$ counts the number of edges that have one endpoint in S and the other in $V \setminus S$:

$$C(S) = |\{(u,v) \in E : u \in S, v \notin S\}|. \tag{12}$$

The goal of the Max-Cut problem is to find a partition S that maximizes this cut value $C(S)$.

To evaluate Max-Cut on a quantum processor, we must convert this discrete graph-theoretic problem into a physical observable. Quantum devices process physical Hamiltonians represented as operators in Hilbert space; therefore, we must map the graph's cost function into an equivalent spin system whose energy corresponds directly to the cut size (22).

We achieve this by assigning a classical binary spin variable $z_v \in \{+1, -1\}$ to each vertex $v \in V$. Here, $z_v = +1$ indicates that vertex v belongs to subset S, while $z_v = -1$ indicates that v belongs to $V \setminus S$. For any edge $(u,v) \in E$, the product $z_u z_v$ equals -1 if the endpoints lie in opposite subsets (a cut edge) and $+1$ if they lie in the same subset. The quantity $\frac{1-z_uz_v}{2}$ therefore evaluates to 1 for a cut edge and 0 otherwise. Summing over all edges gives the classical cut value:

$$C = \sum_{(u,v)\in E} \frac{1 - z_u z_v}{2}. \tag{13}$$

To transition from the classical cost function to a quantum operator, we map each classical spin variable to a Pauli Z operator acting on the corresponding qubit ($z_v \to Z_v$). This yields the quantum cost Hamiltonian H_C:

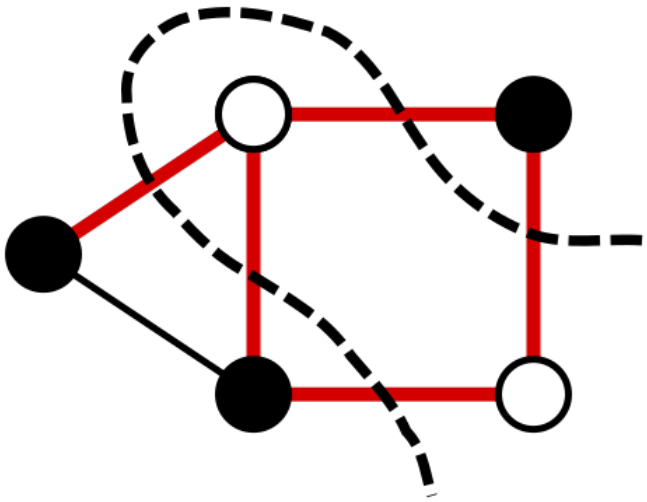

Figure 2.3: Illustration of a Maximum Cut on a sample graph with 5 vertices. The vertices are partitioned into two sets (black and white), and the edges crossing the partition boundary (dotted line) are highlighted in bold red, representing the maximum-cut partition. Figure take from (3).

$$H_C = \sum_{(u,v)\in E} \frac{I - Z_u Z_v}{2}. \tag{14}$$

This operator formulation provides a key advantage: because H_C is composed of commuting two-qubit $Z_u Z_v$ terms, it is diagonal in the computational basis. Every computational basis state $|x\rangle$ (where $x \in \{0,1\}^n$) is an eigenstate of H_C, with its eigenvalue equal to the classical cut size of binary configuration x. The maximum cut size corresponds to the maximum-energy eigenstate of H_C (or equivalently, the ground state of $-H_C$). Preparing a quantum state that maximizes the expectation value $\langle H_C \rangle$ is therefore mathematically equivalent to solving the Max-Cut problem.

To compare the quality of different algorithms across graphs of varying sizes and structures, we require a normalized, instance-independent metric. Raw cut values depend on total edge counts, making direct comparisons difficult. We therefore evaluate algorithms using the approximation ratio r:

$$r = \frac{\langle C \rangle}{C^*}, \tag{15}$$

where $\langle C \rangle = \langle H_C \rangle = \langle \psi \mid H_C \mid \psi \rangle$ is the expected cut value produced by the quantum state $|\psi\rangle$, and C^* is the exact maximum cut value of the graph[2].

Because computing C^* is itself NP-hard, we determine C^* for small benchmark instances ($N \leq 20$) via brute-force classical enumeration over all 2^N configurations, and for larger graphs using an exact integer linear programming solver (Gurobi), as detailed in Chapter 3. The approximation ratio satisfies $0 \leq r \leq 1$, where $r = 1$ indicates that the algorithm has successfully identified an optimal cut configuration.

[2]This defination is only valid for graphs with non-negative weights, alternative defination that is valid for all cases is: $r = \frac{\langle C \rangle - C_{\min}}{C^* - C_{\min}}$)

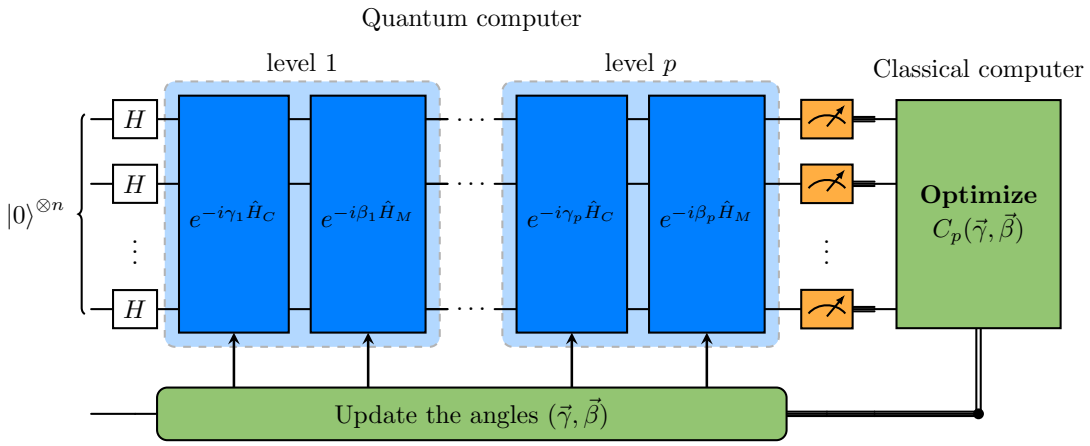

Figure 2.4: The standard QAOA feedback loop. The quantum processor prepares the state $|0\rangle^{\otimes n}$, applies Hadamard gates to create the uniform superposition, evolves it through p alternating cost and mixer layers, and measures the resulting bitstrings. The classical computer estimates the expectation value $\langle H_C \rangle$ from these bitstrings and updates the variational angles γ and β to maximize it. Figure taken from (4).

2.4 Quantum Approximate Optimization Algorithm

Now that we have mapped Max-Cut to a cost Hamiltonian H_C, we need a quantum procedure to prepare states with a large expectation value $\langle H_C \rangle$. The Quantum Approximate Optimization Algorithm (QAOA), introduced by Farhi, Goldstone, and Gutmann (26), achieves this by alternating two operations: a cost unitary that encodes the problem, and a mixer unitary that drives exploration of the solution space.

For Max-Cut, the cost Hamiltonian H_C is defined in Equation 14. The second ingredient is the mixer Hamiltonian H_M, which prevents the state from becoming trapped in a single basis state. The standard choice for Max-Cut is the transverse-field mixer:

$$H_M = \sum_{v \in V} X_v, \tag{16}$$

where X_v is the Pauli X operator acting on the qubit associated with vertex v. The corresponding unitary operators are

$$U_{C(\gamma)} = e^{-i\gamma H_C}, \quad U_{M(\beta)} = e^{-i\beta H_M}. \tag{17}$$

A depth-p QAOA circuit starts from the all-zero state $|0\rangle^{\otimes n}$, applies a Hadamard gate to every qubit to produce the uniform superposition

$$|+\rangle^{\otimes n} = 2^{-\frac{n}{2}} \sum_{x \in \{0,1\}^n} |x\rangle, \tag{18}$$

and then applies p alternating layers of cost and mixer unitaries. The resulting variational state is

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = e^{-i\beta_p H_M} e^{-i\gamma_p H_C} \ldots e^{-i\beta_1 H_M} e^{-i\gamma_1 H_C} \; |+\rangle^{\otimes n}, \tag{19}$$

where $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ are the $2p$ real variational parameters.

Because H_C is diagonal in the computational basis, measuring the final state yields an n-bit string $x \in \{0,1\}^n$ with probability $P(x) = |\langle x \mid \boldsymbol{\gamma}, \boldsymbol{\beta}\rangle|^2$. Repeating this procedure many times allows us to estimate the expectation value of the cost Hamiltonian,

$$\langle C \rangle = \langle \boldsymbol{\gamma}, \boldsymbol{\beta} \mid H_C \mid \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle, \tag{20}$$

which is precisely the expected cut value produced by the circuit. A classical optimizer then updates $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ to maximize this quantity, and the updated parameters are fed back into the quantum circuit. This loop continues until a good set of angles is found Figure 2.4.

The cost unitary can be decomposed into commuting two-qubit rotations, one per edge. Since H_C is a sum of edge terms, we have

$$e^{-i\gamma_l H_C} = \prod_{(u,v)\in E} e^{-i\gamma_l \frac{I - Z_u Z_v}{2}} = e^{-i\gamma_l \frac{|E|}{2}} \prod_{(u,v)\in E} e^{+i\gamma_l \frac{Z_u Z_v}{2}}, \tag{21}$$

where the global phase $e^{-i\gamma_l \frac{|E|}{2}}$ has no observable effect. Each remaining two-qubit term is implemented as a ZZ rotation, which could be decomposed into two CNOT gates and a single-qubit R_Z rotation. The mixer unitary decomposes similarly into independent single-qubit X rotations, one per vertex.

For Max-Cut on unweighted graphs, the eigenvalues of H_C are integers (representing cut sizes), which makes the cost unitary periodic in γ with a period of 2π. The mixer unitary is similarly periodic in β with a period of π (26). We can therefore restrict the initial search space to $\gamma \in [-\pi, \pi)$ and $\beta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)$.

Going beyond simple periodicity, the parameter landscape exhibits deeper symmetries under ZZ -time-reversal and X-parity operations (37). For cycle-free, unweighted regular graphs, these symmetries allow us to fold the parameter domains even further, restricting the search space to a much smaller equivalent region: $\gamma \in \left[0, \frac{\pi}{2}\right]$ and $\beta \in \left[0, \frac{\pi}{4}\right]$. While these tighter bounds are mathematically provable only at depths $p = 1$ and $p = 2$ on cycle-free regular graphs, they hold remarkably well empirically on more general graphs and at higher depths (38). Throughout this thesis, we enforce these folded domains as standard box constraints during optimization.

This domain folding is important for our subgraph transfer strategy. In the transfer pipeline, we assign an angle to a node by averaging the optimal β parameters of all adjacent edges that share that node, from our library. Because the cost and mixer unitaries are symmetric, the same physical state can be described by many different, mathematically equivalent angles. If we do not restrict these parameters to a single convention, we risk storing equivalent but differently signed angles in the library. For example, averaging two physically equivalent mixer angles like $\beta = 0.1$ and $\beta = -0.1$ would yield 0, causing a cancellation that ruins the rotation, even though both angles describe the same physical transformation. Folding every parameter into the standard $\gamma \in \left[0, \frac{\pi}{2}\right]$ and $\beta \in \left[0, \frac{\pi}{4}\right]$ region before it enters the library ensures that averages are taken over comparable values, preventing such destructive interference during transfer.

It is useful to keep in mind what QAOA can provably achieve at small depth. Farhi et al. (26) proved that QAOA with depth $p = 1$ achieves an approximation ratio $r \geq 0.6924$ on 3-regular Max-Cut, and Wurtz and Love (38) extended this to $r \geq 0.7559$ at $p = 2$ and (conjecturally, with strong numerical evidence) $r \geq 0.7924$ at $p = 3$, using fixed angles derived from tree subgraphs (We have used this technique under the name: `Fixed QAOA`). These bounds approach but do not reach the Goemans–Williamson ratio of 0.878. Wurtz and Love further show that no provable quantum advantage over classical heuristics is possible for $p < 6$ on d-regular graphs. The depths $p = 2, 3$ studied in this thesis therefore cannot beat the best classical algorithms in a rigorous sense; rather, we compare the proposed transfer strategy against state-of-the-art quantum algorithms.

2.5 Multi-Angle QAOA

In standard QAOA, every two-qubit cost gate in layer l shares the exact same angle γ_l, and every single-qubit mixer gate shares β_l. While this layer-wide parameter sharing keeps the classical parameter space small ($2p$ variables), it also forces every edge and node in the graph to be treated identically, regardless of their local structural differences. This limits the expressivity of the parameterised state, and caps the achievable approximation ratio at low depths. To overcome this limitation and allow the quantum state to explore a richer subspace of the Hilbert space, Herrman et al. (28) introduced multi-angle QAOA (ma-QAOA). By lifting the uniform-angle constraint, ma-QAOA assigns an independent rotation angle $\gamma_{e,l}$ to each edge $e \in E$ and an independent mixing angle $\beta_{v,l}$ to each vertex $v \in V$ in every layer l.

Collecting these parameters into layer-dependent vectors $\boldsymbol{\gamma}_l = \left(\gamma_{e,l}\right)_{e\in E}$ and $\boldsymbol{\beta}_l = \left(\beta_{v,l}\right)_{v\in V}$, the ma-QAOA state is defined as

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = e^{-iH_M(\boldsymbol{\beta}_p)}e^{-iH_C(\boldsymbol{\gamma}_p)}\ldots e^{-iH_M(\boldsymbol{\beta}_1)}e^{-iH_C(\boldsymbol{\gamma}_1)} \; |+\rangle^{\otimes n}, \tag{22}$$

where the parameterized cost and mixer Hamiltonians are given by

$$H_C(\boldsymbol{\gamma}_l) = \sum_{(u,v)\in E} \gamma_{(u,v),l} \frac{I - Z_u Z_v}{2}, \quad H_M(\boldsymbol{\beta}_l) = \sum_{v\in V} \beta_{v,l} X_v. \tag{23}$$

Each cost gate $e^{-i\left(\frac{\gamma_{(u,v),l}}{2}\right)Z_uZ_v}$ and mixer gate $e^{-i\beta_{v,l}X_v}$ now rotates by its own dedicated angle. Standard QAOA is recovered as a special case when all edge angles in layer l collapse to γ_l and all node angles collapse to β_l. Crucially, standard QAOA and ma-QAOA run on the exact same physical quantum hardware with identical circuit depths and gate counts as can be seen in Figure 2.5; the difference lies entirely in the classical freedom used to parameterize the circuit.

This additional expressivity offers a significant practical advantage for NISQ devices. Across a wide range of graph instances, numerical simulations, and hardware experiments, ma-QAOA has been shown to achieve substantially higher approximation ratios than standard QAOA at the same circuit depth p (28–30). By pushing high solution quality down to shallower depths, ma-QAOA reduces total gate counts, thereby mitigating errors from decoherence and gate infidelities.

However, this expressivity comes at the cost of a severe parameter explosion. A depth-p ma-QAOA circuit requires $p(|E| + |V|)$ classical parameters to optimize. For a d-regular graph, the number of edges is $|E| = d\frac{N}{2}$. On 3-regular graphs ($d = 3$), the parameter count becomes

$$p\left(\frac{3N}{2} + N\right) = \frac{5}{2}pN, \tag{24}$$

which grows linearly with both system size N and circuit depth p. For example, an $N = 18$ graph at depth $p = 2$ requires 90 parameters, rising to 135 parameters at $p = 3$.

This parameter growth severely worsens the classical trainability challenges that affect near-term variational algorithms. As discussed in Chapter 1.2, the two primary bottlenecks of VQAs are classical optimization complexity and barren plateaus, which are already present in standard QAOA, but they become much more severe under the ma-QAOA. Firstly, moving from standard QAOA's $2p$ parameters to ma-QAOA's $O(pN)$ independent angles expands the classical search space into a higher-dimensional landscape. This landscape is riddled with local minima and saddle points. Classical gradient-based optimisers require far more circuit evaluations to calculate gradients and navigate this space, and they are highly prone to getting trapped in suboptimal local minima. This makes the classical training step computationally expensive and heavily dependent on a large number of random restarts (32). Secondly, in standard QAOA, the parameter count is small and

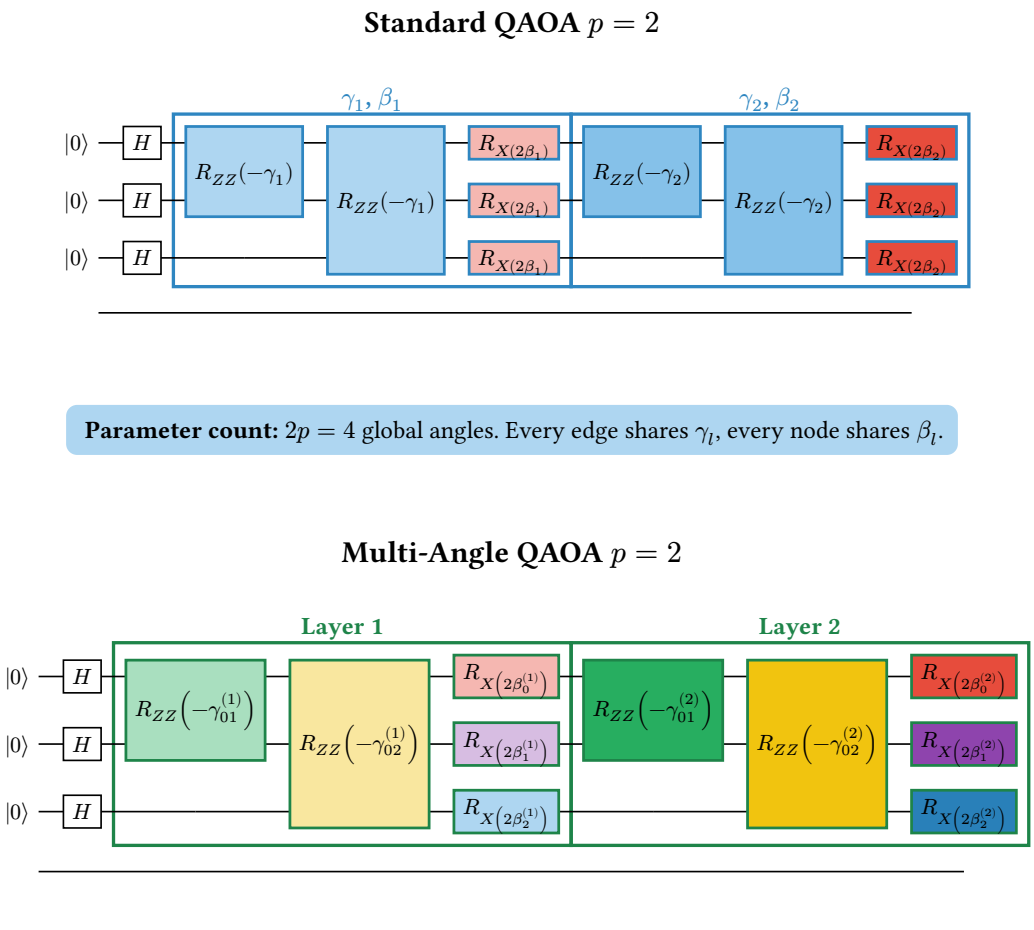

Figure 2.5: Comparison of standard QAOA (top) and multi-angle QAOA (bottom) at depth $p = 2$ on a 3-qubit system. Standard QAOA uses $2p = 4$ global parameters shared across all gates. Ma-QAOA assigns an independent parameter to every edge ($\gamma_{e,l}$) and every node ($\beta_{v,l}$), yielding $p(|E| + |V|) = 10$ parameters. Both algorithms execute on the exact same quantum hardware; the extra expressivity comes entirely from the classical parameterization.

independent of system size, which keeps the barren plateau issue somewhat manageable at low depths. In ma-QAOA, however, the parameter count grows with the size of the physical system. This parameter growth exponentially expands the dimension of the generated dynamical Lie algebra, which mathematically guarantees that the gradients will vanish exponentially across almost the entire optimization landscape (39). Consequently, as the graph size N increases, gradient-based classical training of ma-QAOA from scratch becomes practically impossible. A rigorous algebraic derivation of this connection between the parameter count, the Lie algebra of the gates, and the exponential flattening of the landscape is detailed in Appendix B.

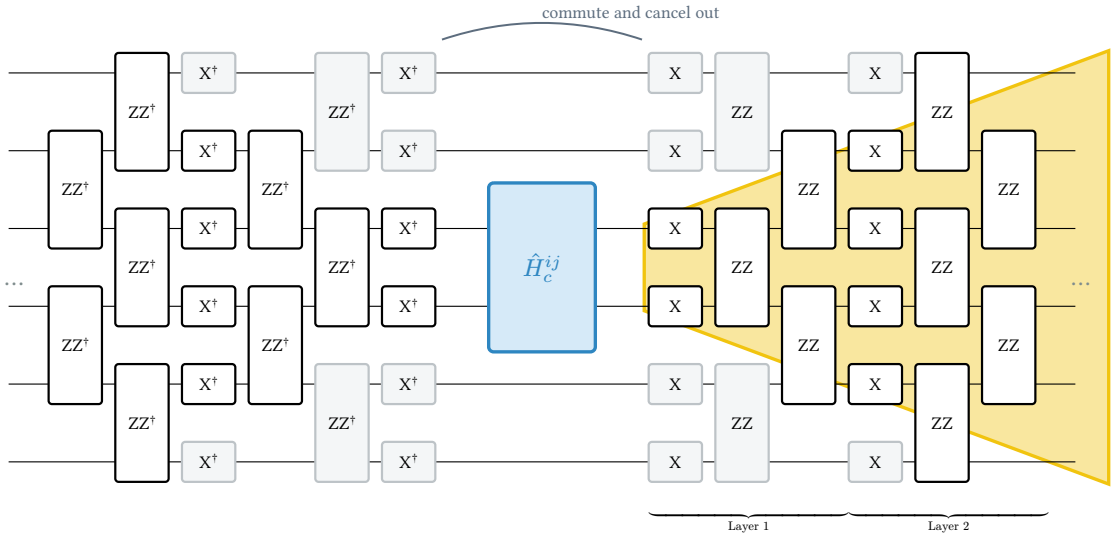

Figure 2.6: Reverse causal cone for edge (i, j) at depth p. The operator support grows by at most one graph step per layer, and only gates inside the cone contribute to the edge expectation. Gates outside the cone (gray) commute with the operator and cancel.

2.6 Light-Cone Locality

In this section we study the locality property of QAOA and ma-QAOA, which motivates one of the parameter-setting strategy we have used in our thesis. Understanding this property requires the concept of the reverse causal cone, first discussed in the context of QAOA by Farhi, Goldstone, and Gutmann (26).

We established in Equation 14 that the cost hamiltonion for the whole graph, can be written as the summation of cost hamiltonion of each edge, so we write

$$H_C = \sum_{e \in E} H_e, \quad H_e = \frac{I - Z_u Z_v}{2} \quad \text{for } e = (u, v), \tag{25}$$

and by linearity of expectation value, $\langle C \rangle = \sum_{e \in E} \langle H_e \rangle$. This decomposition is exact, so we have done no approximation when we analyse the total cut value one edge at a time.

Next, instead of evolving the state forward through the circuit, we evolve the operator backward. Writing the full circuit as $U = U_M(\beta_p)U_C(\gamma_p) \cdots U_M(\beta_1)U_C(\gamma_1)$, the expectation value of a single edge operator becomes

$$\langle H_e \rangle = \langle + |^{\otimes n} \, U^\dagger H_e U \, | + \rangle^{\otimes n} = \langle + |^{\otimes n} \, \tilde{H}_e^{(p)} \, | + \rangle^{\otimes n}, \tag{26}$$

where $\tilde{H}_e^{(p)} := U^\dagger H_e U$ is the effective operator acting on the initial superposition state. We call the set of qubits on which $\tilde{H}_e^{(p)}$ acts non-trivially its support, $\mathrm{supp}\left(\tilde{H}_e^{(p)}\right)$. Any gate that acts entirely outside this support commutes with the operator and cancels with its adjoint ($U^\dagger U = I$), so qubits outside the support contribute nothing to $\langle H_e \rangle$. This is visually represented in the Figure 2.6. However, we shall address how large this support becomes, upon one iteration through the QAOA circuit.

This follows from how the two types of gates that act on the operator as it is propagated backward layer by layer (the explicit commutator calculations are collected in Appendix A). The mixer unitary

$U_M(\beta) = \prod_{w \in V} e^{-i\beta X_w}$ consists of single-qubit rotations[3]. On a qubit inside the support it only rotates the operator, for example

$$e^{i\beta X_w} Z_w e^{-i\beta X_w} = \cos(2\beta) Z_w + \sin(2\beta) Y_w, \tag{27}$$

so the mixer changes the operator locally but does not enlarges its support. The cost unitary $U_C(\gamma) = \prod_{(a,b)\in E} e^{-i\gamma H_{(a,b)}}$ behaves differently. A cost gate whose edge is disjoint from the support commutes and cancels out; a gate whose both endpoints lie inside the support leaves the support unchanged; but a gate with one endpoint inside the support and one endpoint outside increases the size of support. For example, Y_w is transformed into a combination containing $X_w Z_x$, where x is the neighbouring qubit outside. Only in this boundary case does the support grow, and it grows by exactly one vertex.

So starting from $\mathrm{supp}(H_e) = \{u, v\}$ and applying p layers, the support can therefore expand by at most one graph step per layer. The effective operator is then bounded by a ball of radius p around the central edge:

$$\mathrm{supp}\left(\tilde{H}_e^{(p)}\right) \subseteq B_p(e), \tag{28}$$

where $B_p(e)$ denotes the set of vertices within graph distance p of the edge e, i.e, a p-depth subgraph with e as central edge. The region containing all gates that do not cancel is called the **reverse causal cone** of the edge, and the argument above shows that it is always contained in the p-depth subgraph induced by $B_p(e)$. The formal derivation of this containment is given in Appendix A.

On a d-regular graph, the ball $B_p(e)$ contains at most $2\sum_{k=0}^{p}(d-1)^k$ vertices in the worst case of a tree-like neighbourhood(a graph with no cycles), which scales as $O((d-1)^p)$ and is independent of the total graph size N. In most practical graphs, short cycles inside the ball merge branches of this tree, so the actual subgraphs are typically smaller than this bound.

The consequence that we follow from this locality structure is that the per-edge expectation $\langle H_e \rangle$ depends only on the p-depth subgraph around e and on the angles assigned inside it, and hence we focus our effort on setting the parameters inside this smaller subgraph instead of larger host graph. This is what our subgraph library precomputes (Chapter 3), and transfer to full host graph.

2.7 Previous Works

Optimising standard QAOA and multi-angle ma-QAOA from a cold, random start becomes computationally expensive as the graph size grows. This is both because each evaluation of the cost Hamiltonian requires a full quantum state simulation and because the high-dimensional, non-convex landscapes discussed in Section 2.5 and Appendix B require numerous random restarts to identify high-quality minima. To bypass or mitigate this classical training bottleneck, several strategies parallel to ours have been proposed in the literature. Our subgraph-library transfer and slope-constrained optimization strategies are built directly on the foundations laid by these works. In this section, we review some of these previous works, dividing the literature into a few broad categories based on the specific aspects of our work they motivate.

2.7.1 Parameter Concentration and Reusability

The fundamental justification for transferring variational parameters across different instances of a problem comes from the "concentration of measure" phenomenon. Brandão et al. (40) demonstrated that for typical problem instances of a given class and size, the optimal QAOA parameters are highly concentrated. Specifically, when fixed control parameters are applied to typical instances,

[3] All the calculations are done with respect to Cost and mixer hamiltonions for the Max-cut problem.

the resulting expectation values cluster tightly around a single expected value. This implies that parameters optimized for a single typical instance can be directly reused on new, unseen instances of the same class without undergoing a new, computationally expensive classical optimization loop.

Authors also gave a similar parameter-reusability argument when scaling the circuit depth p. The parameters that produce a high-quality optimization at lower depths can serve as a robust starting point for optimization at larger depths. They explain it with an example, to solve Max-Cut on a graph of size $N = 100$ at depth $p = 6$, one can first optimize the parameters on a smaller instance with $N = 20$ qubits. These optimized parameters can then be transferred to a medium-sized instance of $N = 50$ qubits and slightly refined, before finally being applied to the target $N = 100$ instance with a final minor refinement step. This bootstrap-like approach demonstrates that optimal parameters concentrate predictably both across distinct problem instances and across varying circuit depths.

Building on this, Montañez-Barrera et al. (41) explored transferring optimal standard QAOA parameters across different problem instances and even across entirely different problem classes. By training parameters on small instances of various combinatorial problems—including the travelling salesperson problem, bin packing, knapsack, and portfolio optimization—they showed that parameters optimized for certain problems, such as bin packing, retain high performance when transferred to larger instances and different problem domains. While their work focuses on transferring standard QAOA parameters across different problem classes, our work keeps the problem class fixed (Max-Cut) and instead transfers parameters across different circuit structures (from small QAOA subgraphs to full ma-QAOA circuits).

Symmetry also plays a key role in parameter reusability. Lyngfelt et al. (37) studied how symmetries in the QAOA cost landscape create families of mathematically equivalent optimal parameters, and identified which of these parameter sets transfer successfully between graph instances. This work establishes that edges sharing the same local structure can share the same angles, provided the angles are folded into a consistent symmetry domain. We extend this principle to develop our subgraph-transfer strategy.

2.7.2 Local Tree-Like Approximations and Fixed Angles

Farhi et al. (26), in their original paper introducing QAOA, noted that using fixed, pre-optimized parameters could serve as a viable and efficient shortcut to classical optimization. They derived a lower bound of 0.6924 on the approximation ratio for Max-Cut on 3-regular graphs at depth $p = 1$. Wurtz and Love (38) extended this analysis to higher depths, showing that fixed parameters optimized for an infinite regular tree also achieve competitive performance ($r \geq 0.7559$ at $p = 2$ and $r \geq 0.7924$ at $p = 3$), which serves as a lower bound for standard QAOA at these depths. Because regular graphs are locally tree-like at shallow depths p, these tree-derived fixed angles achieve high approximation ratios with zero runtime classical training.

We use these tree-derived fixed angles as a major baseline in our work, verifying that they provide a remarkably robust classical shortcut.

2.7.3 Low-Complexity and Constrained Parameterized States

Because full ma-QAOA training is classically expensive but standard QAOA is structurally restrictive, several works have proposed intermediate-complexity architectures. Chalupnik et al. (42) introduced QAOA+, a modified algorithm that inserts an extra set of problem-independent gates at the end of the standard QAOA layers. They showed that this architecture improves the approximation ratio of standard $p = 1$ QAOA without requiring the deep gates of $p = 2$ QAOA or the massive parameter count of ma-QAOA. Benchmarked on Max-Cut for regular graphs with up to 14 nodes,

QAOA+ achieves intermediate approximation ratios using significantly fewer two-qubit gates and classical parameters, providing an efficient alternative for near-term hardware.

To manage the parameter count of ma-QAOA directly, other strategies constrain how the variational parameters are defined or updated. Jang et al. (43) proposed a layerwise retraining scheme that freezes most ma-QAOA layers and updates only a single layer at a time, cycling through the layers in subsequent iterations to reduce classical optimization overhead. Additionally, Montañez-Barrera et al. (5) studied the "linear ramp" or linear trend observed in optimal standard QAOA parameters across layers, showing that enforcing this linear profile significantly accelerates training.

Our second strategy, slope-constrained ma-QAOA, directly extends this linear ramp concept to the multi-angle parameterized state. While previous works have studied standard QAOA parameter concentration, cross-instance transfer learning, and ma-QAOA layer freezing, the systematic transfer of parameters from a precomputed subgraph library trained on QAOA into whole-graph ma-QAOA, alongside linear-ramp-based slope constraints, has not been systematically explored. This is the gap that the present thesis aims to fill.

Chapter 3

Methods and Simulation Setup

This chapter describes the classical simulation setup, graph generation procedures, and numerical optimization protocols used to implement and evaluate our parameter-setting strategies.

3.1 Graph Instances

Throughout this study, we evaluate our algorithms on random 3-regular graphs, in which every node is connected to exactly three neighbors. We select 3-regular graphs because they are the lowest-degree regular graphs that are classically non-trivial to solve. For comparison, degree-2 regular graphs are simple collections of disjoint cycles (rings), which can be solved easily by assigning alternating labels to consecutive vertices. In contrast, degree-3 regular graphs already exhibit rich structural variation, making them the standard benchmark in both classical and quantum approximation literature (26,38).

We restrict our study to even vertex counts N because a d-regular graph contains $d\frac{N}{2}$ edges, a quantity that must be an integer. For $d = 3$, the number of edges is $3\frac{N}{2}$, which is an integer if and only if N is even. We evaluate and present data for graph sizes $N \in \{8, 10, 12, 14, 16, 18, 20\}$. To obtain statistically significant results, we generate 20 independent graph instances (seeds) for each size N. Every data point presented in our results represents the expectation value or approximation ratio averaged over these 20 independent graph seeds.

To compute the approximation ratio r defined in Equation 15, we require the exact maximum cut value C^*. Since our system sizes are within the classically simulable regime ($N \leq 20$), we obtain C^* exactly via brute-force classical enumeration over all 2^N possible binary bitstrings.

3.2 Subgraph Transfer Strategy

Our first parameter-setting strategy is the **Subgraph Parameter Transfer** pipeline. The core idea is to completely bypass the classical optimization loop on large target graphs by assigning precomputed, optimal variational parameters directly to the gates. This pipeline consists of three main steps:

1. Constructing a complete library of all possible unique local p-depth subgraphs.
2. Optimizing the variational parameters for the central edge of each unique subgraph in the library on QAOA circuit.
3. Matching each edge of a new target graph to its corresponding library entry, transferring the pre-optimized edge parameters, and averaging the transferred node parameters.

Once the parameters are assigned, we run the ma-QAOA circuit on the target graph only once, avoiding classical parameter training entirely.

3.2.1 Subgraph Library Construction

The first step in the transfer pipeline is the construction of the subgraph library. Building this library is a one-time classical precomputation that is not included in the runtime evaluation of the quantum circuit. Once the library is built, it is stored and reused for all subsequent transfer runs.

Since we are focusing on 3-regular graphs, we construct a library of all possible unique 3-regular local subgraphs at depths $p = 1$ and $p = 2$. The p-depth neighborhood of a central edge $e = (u, v)$ contains all vertices and edges within a graph distance p from either u or v. As shown in Section 2.6 and Appendix A, the expectation value $\langle H_e \rangle$ (Equation 26) of a central edge depends only on the gates within this local p-depth light cone (or environment). Therefore, the library must contain every possible local p-depth neighborhood environment that an edge can encounter in any 3-regular host graph.

A p-depth, d-regular subgraph is a graph where the distance from any vertex to the central root edge is at most p, and all vertices, except for the boundary "leaf" nodes at distance exactly p, must satisfy the degree requirement d. To construct this library, we adopt a recursive generation strategy inspired by Wurtz and Love (38). Instead of generating subgraphs from scratch, we build the depth-p library by incrementally growing the unique subgraphs already stored in the $(p-1)$-depth library.

For the $p = 1$ library, we initialize the generation with a single root edge (two vertices connected by an edge). For higher depths, the starting points are the subgraphs of the preceding depth. We then grow each subgraph iteratively using the following steps until no further vertices or edges can be added:

- First step is to identify all "open" vertices, which are vertices whose current degree is less than 3 and whose distance from the central root edge is strictly less than p. An open vertex represents an unfinished boundary that would receive additional connections in a full 3-regular host graph. If a subgraph has no open vertices, it is considered complete and is added to the candidate list.
- We select the open vertex with the smallest numerical index and extend the graph. Done either by connecting it to another open vertex already in the subgraph (which closes a local cycle) or by attaching a new vertex using one, two, or three links. Because the host graph is 3-regular, no vertex can have more than three incident edges.
- Any boundary vertex located at a distance of exactly p from the root edge is never linked to another boundary vertex. An edge connecting two boundary vertices lies outside the depth-p light cone of the central edge (Appendix A) and therefore cannot influence the expectation value $\langle H_e \rangle$.

Because every valid p-depth neighborhood can be constructed by resolving the open vertices of its $(p-1)$-depth restriction one by one, this procedure is guaranteed to generate all possible 3-regular subgraphs up to rooted isomorphism.

To ensure the library contains only unique structures, once we have a newly generated candidate subgraph, it is compared against the existing library entries using a Weisfeiler-Lehman (WL) hash and an exact rooted isomorphism check, as detailed in Appendix C.

Running this generation pipeline yields exactly 3 unique subgraphs at depth $p = 1$ (corresponding to the central root edge having zero, one, or two triangles passing through it) and 123 unique subgraphs at depth $p = 2$. At depth $p = 3$, the number of unique subgraphs grows to 913,088. These counts are in exact agreement with the combinatorial results of Wurtz and Love (38). This rapid growth shows why maintaining an explicit depth-3 library is computationally infeasible, motivating the parameter extrapolation techniques discussed in later sections.

Once the unique subgraphs are identified, we compute the optimal standard QAOA parameters (γ, β) that maximize the expected value of the central root edge $\langle H_e \rangle$ (Equation 26) for each subgraph. We perform the classical optimization step using the Sequential Least Squares Programming (SLSQP) optimizer with 15 independent random restarts and simulate the quantum state exactly with Qiskit's `Statevector` simulator. During optimization, we restrict the parameters to the folded symmetry domains $\gamma \in [0, \frac{\pi}{2}]$ and $\beta \in [0, \frac{\pi}{4}]$ (Section 2.4) to ensure consistent parameter conventions across all library entries.

3.2.2 Transfer Pipeline

Once the subgraph library has been constructed and the variational parameters optimized for each unique entry, the next step is to map these precomputed parameters onto a new, larger target graph. We now describe the transfer pipeline step by step. Given a new target graph G and a target depth p, the parameter mapping proceeds as follows:

- Extract the p-depth Subgraphs: For every edge $e \in E(G)$, we extract its local p-depth neighborhood subgraph G_e using a breadth-first search (BFS) starting simultaneously from both of the edge's endpoints. This BFS identifies all vertices within a graph distance of p or less from either endpoint, and we include all edges connecting these identified vertices.
- Match to Library Entries: We compute the rooted WL hash of G_e and look up the corresponding hash in our subgraph library. In the case of a hash collision, we perform a rooted isomorphism check to confirm the structural match, using the same procedure we used to ensure uniqueness during library construction (Appendix C). This matching returns the unique library subgraph ID along with its pre-optimized angles. Because our library construction ensured to contain all possible 3-regular local neighborhood structure, every edge in a 3-regular target graph is guaranteed to find a valid match.
- Transfer Edge Parameters: We assign the pre-optimized library angles directly to the target edge. Specifically, in each layer l, the target edge e receives the edge parameter $\gamma_{e,l}$ equal to the library's optimized γ_l, and its two endpoints receive the library's optimized β_l value.
- Perform Node Averaging: Since each vertex v in a graph is common to multiple edges, it receives a different β_l candidate from each of those incident edges. To resolve these multiple assignments, we compute the final parameter for each node v by taking the mean of the β_l values it received from its incident edges. This node-averaging step resolves conflicts while preserving the local structural context of the node's neighborhood, yielding exactly one β_v per node and one γ_e per edge in every layer. This step is made possible because we restricted the optimized parameters in the library to be in a single positive domain Section 2.4.
- Execute Circuit Simulation: We construct the complete ma-QAOA circuit using these assigned per-edge and per-node angles and simulate it once, evaluating the whole-graph expected cut value $\langle C \rangle$ from the final state.

At no point in this pipeline do we run a classical optimization loop on the full graph. Classically, we ran the BFS subgraph generation, library search and parameter allocation. The quantum resource required is a single circuit evaluation, which is carried out using Qiskit's Statevector simulator. Because the pipeline requires a precomputed library at the same target depth, this direct transfer applies to depths $p \in \{1, 2\}$. For depth $p = 3$, we employ the parameter extrapolation techniques described in the next section to bypass the exponential library size explosion. The entire transfer pipeline is illustrated in Figure 3.1.

Figure 3.1: The transfer pipeline discussed in Chapter 3.2.2. A full graph G is decomposed into per-edge p-depth BFS subgraphs. Each subgraph is matched to a pre-optimized library via WL hashing and rooted isomorphism. The library edge angles γ_e are assigned directly, while node angles β_v are obtained by averaging the candidate values received from all incident edges. Finally, the whole-graph expectation value $\langle C \rangle$ is evaluated via classical statevector simulation.

3.2.3 Variants of the Subgraph-Transfer Strategy

The above discussed transfer pipeline assigns both the cost and mixer angles from the library and evaluates the circuit once, with no optimization loop. We also tested three related variants of this strategy.

First one is termed `extrapolation p3`, where we use linear ramp protocol (Section 3.3.1) to estimate $p = 3$ using subgraph library at $p = 2$, bypassing the need to generate a explicit $p = 3$ subgraph library. This method is detailed in Section 3.3.3.

Second variant is what we have named semi-transfer strategy. In this we transfer only one of the two parameter families and optimize the other on the full target graph. This tests whether the cost angles or the mixer angles are more sensitive to the whole-graph structure. Further, we have two sub-variants in this:

- `transfer` β_{lib} `+ opt` γ: We assign the node mixer angles β_v by averaging the library values over the incident edges of each vertex, as in the full transfer pipeline. All edge cost angles γ_e are then treated as free variables and optimized layer by layer. This leaves $p\ |E|$ active parameters.
- `transfer` γ_{lib} `+ opt` β: We assign the edge cost angles γ_e directly from the matched library entries and optimize all node mixer angles β_v layer by layer. This leaves $p\ |V|$ active parameters.

Both variants reduce the number of optimized variables compared with full ma-QAOA, however they do still require a classical optimization loop on the target graph.

A third variant uses the transferred parameters as the starting point for a full ma-QAOA optimization, rather than as a final assignment(`warm1_maqaoa_p2` and `warm1_maqaoa_p3`). For $p = 2$ we initialize the full ma-QAOA circuit with the transferred subgraph parameters. For $p = 3$ we initialize with the extrapolated parameters from the linear-ramp protocol(Section 3.3.3). We then run a single SLSQP optimization from this warm start, instead of multiple random restarts used in the standard ma-QAOA benchmarks. We study if the trasnferred library parameters act as a good initial guess that can reduce the classical optimization effort while still reaching a competitive approximation ratio.

3.3 Linear-Ramp Slope Optimization and Extrapolation

In this section, we introduce our second strategy which is based on the linear-ramp protocol. We have utilized linear-ramp protocol in two distinct ways: first, as an optimization constraint on the full-graph ma-QAOA circuit to reduce the number of optimizable parameters, and second, as an extrapolation rule to scale our $p = 2$ library to $p = 3$ parameters without maintaining and training an explicit library for $p = 3$.

3.3.1 The Linear-Ramp Protocol

The linear-ramp protocol originates from empirical observations that optimal variational parameters (γ_l, β_l) in standard QAOA often follow a smooth, linear trajectory across successive layers (5,44). In this protocol, the cost angles increase linearly from layer to layer, while the mixer angles decrease linearly. Mathematically, for a circuit of depth p, the parameters in layer $l \in \{0, ..., p-1\}$ are parameterized by two slope variables, Δ_γ and Δ_β:

$$\gamma_l = \frac{l+1}{p}\Delta_\gamma, \quad \beta_l = \left(1 - \frac{l}{p}\right)\Delta_\beta. \tag{29}$$

Under this schedule, the cost rotation angle starts at $\frac{1}{p}\Delta_\gamma$ in the first layer and increases to Δ_γ in the final layer. Conversely, the mixing rotation angle starts at Δ_β in the first layer and shrinks to $\frac{1}{p}\Delta_\beta$ in the final layer, as illustrated in Figure 3.2.

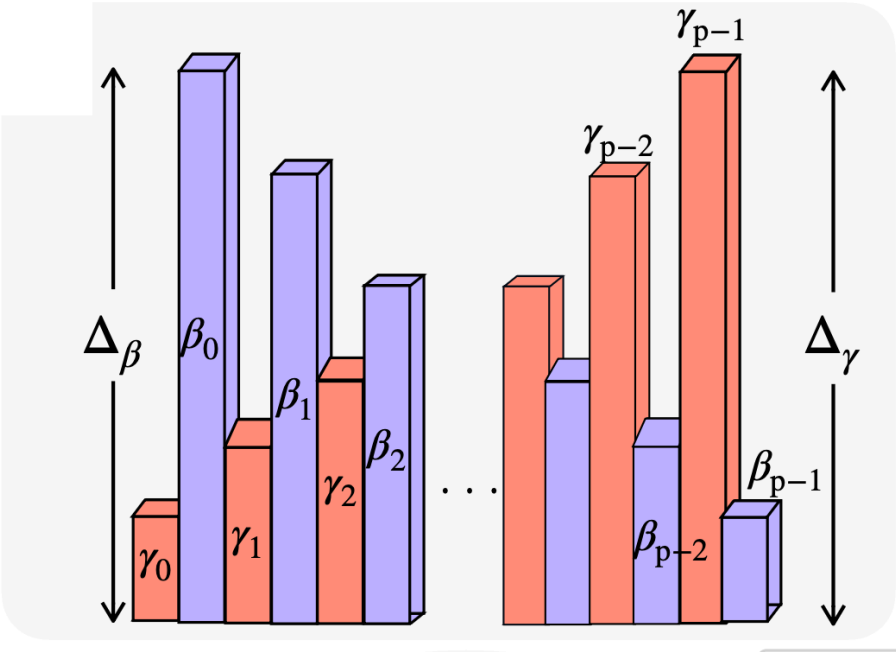

Figure 3.2: The Linear Ramp Protocol. The cost parameters γ_l (blue) increase linearly from layer to layer, while the mixer parameters β_l (red) decrease linearly. Figure taken from (5).

3.3.2 Slope-Constrained ma-QAOA

We extend this linear-ramp concept to the multi-angle parameterization by applying the constraint locally to every edge and node. Instead of allowing all $p(|E| + |V|)$ gate parameters to vary independently, we constrain the parameters of each individual edge $e \in E$ and node $v \in V$ to follow their own linear slopes across the p layers. For each layer $l \in \{0, ..., p-1\}$, the multi-angle parameters are defined as:

$$\gamma_{e,l} = \frac{l+1}{p}\Delta_{\gamma_e}, \quad \beta_{v,l} = \left(1 - \frac{l}{p}\right)\Delta_{\beta_v}, \tag{30}$$

where Δ_{γ_e} is the cost slope associated with edge e, and Δ_{β_v} is the mixing slope associated with vertex v. By imposing this constraint, we reduce the number of active optimization parameters from $p(|E| + |V|)$ to exactly $|E| + |V|$ parameters. Crucially, this parameter count remains constant regardless

of the circuit depth p. This reduction in dimensionality simplifies the non-convex optimization landscape and reduces the possibility of barren-plateau effects, facilitating faster classical convergence while still allowing for localized, structural variations across the graph. We refer to this method as slope-constrained ma-QAOA (labeled as `ma_qaoa_slope_opt` in our benchmarks).

3.3.3 Parameter Extrapolation to Depth $p = 3$

We have also utilized linear-ramp protocol as an extrapolation rule to scale our subgraph-matching pipeline to $p = 3$ without constructing a $p = 3$ library. For each unique subgraph in our precomputed $p = 2$ library, we have already obtained the optimized standard QAOA parameters $\gamma = (\gamma_1, \gamma_2)$ and $\beta = (\beta_1, \beta_2)$. Under the linear-ramp assumption of Equation 29, we can extract the underlying slopes Δ_γ and Δ_β from these optimized $p = 2$ parameters:

- For the cost parameters, the linear-ramp schedule at $p = 2$ implies $\gamma_1 = \frac{1}{2}\Delta_\gamma$ and $\gamma_2 = \Delta_\gamma$. This provides two independent estimates of the cost slope: $\Delta_\gamma^{(1)} = 2\gamma_1$ and $\Delta_\gamma^{(2)} = \gamma_2$. We take the average of these estimates to define the final cost slope for the subgraph:
$$\Delta_\gamma = \frac{2\gamma_1 + \gamma_2}{2}. \tag{31}$$
- For the mixing parameters, the schedule at $p = 2$ implies $\beta_1 = \Delta_\beta$ and $\beta_2 = \frac{1}{2}\Delta_\beta$. This similarly yields two estimates of the mixer slope: $\Delta_\beta^{(1)} = \beta_1$ and $\Delta_\beta^{(2)} = 2\beta_2$. We average these to define the final mixer slope:
$$\Delta_\beta = \frac{\beta_1 + 2\beta_2}{2}. \tag{32}$$

Once we have computed the slopes Δ_γ and Δ_β for each of the 123 entries in our $p = 2$ library, we can extrapolate these parameters to $p = 3$ using Equation 29 with $p = 3$. This yields three layer-wise parameters for each library entry:

$$\gamma_1 = \frac{1}{3}\Delta_\gamma, \quad \gamma_2 = \frac{2}{3}\Delta_\gamma, \quad \gamma_3 = \Delta_\gamma, \tag{33}$$

$$\beta_1 = \Delta_\beta, \quad \beta_2 = \frac{2}{3}\Delta_\beta, \quad \beta_3 = \frac{1}{3}\Delta_\beta. \tag{34}$$

This extrapolation allows us to assign depth-3 parameters to our target graphs dynamically without requiring a physical depth-3 library. During transfer, we match each edge of the target graph to its $p = 2$ neighborhood subgraph in our library, compute the extrapolated $p = 3$ parameters, and apply the standard transfer pipeline. We refer to this approach as the parameter extrapolation strategy (labeled as extrapolate_p3 in our results).

3.4 Optimization Protocols

We now summarize all the methods compared in this thesis. For each method we list the circuit depth, the total number of variational parameters, whether those parameters are optimized on the target graph, and where they come from. All methods are collected in Table 3.1.

Method	p	#Parameters	Optimization	Source
QAOA opt (Section 2.4)	2, 3	$2p$	Yes	Full-graph SLSQP
ma-QAOA opt (Section 2.5)	2, 3	$p(\|E\| + \|V\|)$	Yes	Full-graph SLSQP
Fixed QAOA (Section 2.4)	2, 3	$2p$	No	Tree-subgraph parameters
Transfer (p = 2) (Section 3.2.2)	2	$p(\|E\| + \|V\|)$	No	Subgraph library
Extrapolate (p = 3) (Section 3.3.3)	3	$p(\|E\| + \|V\|)$	No	Linear ramp using p = 2 library
ma-QAOA slope opt (Section 3.3.2)	2, 3	$\|E\| + \|V\|$	Yes	Full-graph SLSQP on linear ramp
Transfer γ_{lib}, optimize β (Section 3.2.3)	2	$p(\|E\| + \|V\|)$	Partial	$p\|E\|$ from library, $p\|V\|$ optimized
Transfer β_{lib}, optimize γ (Section 3.2.3)	2	$p(\|E\| + \|V\|)$	Partial	$p\|V\|$ from library, $p\|E\|$ optimized
Warm ma-QAOA (Section 3.2.3)	2, 3	$p(\|E\| + \|V\|)$	Yes	Warm-start SLSQP with transfer parameters

Table 3.1: Summary of the methods compared in this thesis. The columns give the circuit depth(p), the total number of variational parameters, whether classical optimization is run on the target graph, and where the parameters come from.

3.5 Simulation Backends

Every method described in the previous section simulates the same depth-p quantum circuit, differing only in its variational parameters and how those parameters are obtained. We prepare a depth-p QAOA or ma-QAOA state and compute the expectation value $\langle C \rangle$ of the Max-Cut cost Hamiltonian. On real hardware this would be done by running the circuit and measuring the resulting bitstrings in the computational basis. Here we simulate the circuit classically, evaluating the exact expectation value directly from the state vector without sampling. This section describes the gate-level implementation and the simulator setup.

The circuit starts from the all-zero state and applies a Hadamard gate to every qubit, producing the uniform superposition over all 2^N bitstrings (Equation 8). Each layer then applies the cost and mixer unitaries introduced in Section 2.4. Up an additive constant(energy offset), the Max-Cut cost Hamiltonian is

$$H_C = -\frac{1}{2} \sum_{(i,j) \in E} Z_i Z_j, \tag{35}$$

so the edge unitary is

$$e^{-i\gamma(-\frac{1}{2})Z_i Z_j} = R_{ZZ}(-\gamma), \tag{36}$$

where $R_{ZZ}(\theta) = e^{-i\frac{\theta}{2}Z_i Z_j}$ is the two-qubit ZZ rotation gate. The mixer Hamiltonian is $H_M = \sum_i X_i$, so the single-qubit rotation on each qubit is

$$e^{-i\beta X_i} = R_{X(2\beta)}. \tag{37}$$

This provides the exact mapping from variational parameters to the rotation angles applied by the quantum gates.

In standard QAOA the same angles are used on every edge and node in a given layer. In ma-QAOA and its variants each edge and each node receives its own layer-dependent angles, but the underlying gate structure remains identical. The complete circuit is shown in Figure 3.5.

We use Qiskit's `Statevector` simulator for all circuits up to $N \leq 20$. It evolves the full 2^N-dimensional complex vector through the gate sequence and returns the exact expectation value $\langle C \rangle$ without any shot noise ((34,45)). Because the statevector stores 2^N double-precision complex amplitudes (16 bytes per amplitude), its memory requirement grows exponentially. For $N = 20$ this is approximately 16 MB, which fit comfortably within our simulation hardware.

3.6 Hardware and Evaluation Metrics

All simulations were run on a local Apple MacBook Pro with an M5 Pro chip (18 CPU cores, 20 GPU cores, 24 GB RAM)[4].

The main quantity we compare across methods is the approximation ratio r defined in Equation 15, reported as a function of graph size $N = |V|$. Alongside it we also report the optimization wall time(or simply runtime).

Runtime is measured as the elapsed wall-clock time for the full simulation on the target graph, including all SLSQP restarts and, for transfer-based methods, the time to match BFS edge subgraphs against the library. It does not include the one-time construction and optimization of the subgraph library, since that cost is a one time cost over all transfer runs.

Every reported value is an average over 20 independent random 3-regular graph instances (seeds) of the same size N. This averaging reduces instance-to-instance fluctuations and gives a representative trend across graph sizes. Standard deviations are shown where relevant.

[4]Except the full ma-QAOA optimization (`ma_qaoa_opt`) at p=3, which was run on the Titan compute server at Chalmers (15 CPU cores, 2 NVIDIA RTX 2080 Ti GPUs, 24 GB VRAM)

Chapter 4

Results and Discussion

In this chapter, we evaluate the performance of our parameter-setting strategies on random 3-regular graphs across different system sizes N. We study subgraph parameter transfer (Section 3.2.2), in which variational parameters are assigned directly from a pre-optimized library of local p-depth subgraphs without training on the target graph. Secondly, we consider slope-constrained ma-QAOA (Section 3.3.2), in which the multi-angle parameters are restricted to linear layer-wise ramps, reducing the optimization search space to $|E| + |V|$ optimizable parameters.

We structure our evaluation as follows. First, we compare the approximation ratios of subgraph transfer (`transfer_p2` and `extrapolate_p3`) against three baseline methods: standard optimized QAOA (`qaoa_opt`), fully optimized ma-QAOA (`ma_qaoa_opt`), and fixed-angle QAOA (`fixed_qaoa`). Second, we analyze the classical runtime and computational scaling of these strategies. Third, we evaluate hybrid variants, including warm-started ma-QAOA and semi-transfer protocols (Section 3.2.3). Finally, we assess the performance and optimization efficiency of slope-constrained ma-QAOA(`ma_qaoa_slope_opt`).

4.1 Subgraph Transfer Strategy

Figure 4.1 and Figure 4.2 display the approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N for circuit depths $p = 2$ and $p = 3$, respectively. Across all evaluated problem sizes N, fully optimized ma-QAOA (`ma_qaoa_opt`) achieves the highest mean approximation ratio. The remaining three methods, `qaoa_opt`, `fixed_qaoa`, and `transfer_p2` (or `extrapolate_p3` at $p = 3$), tend to cluster within a narrow band. At depth $p = 2$, standard optimized QAOA (`qaoa_opt`) achieves a slightly higher mean approximation ratio than `transfer_p2` and `fixed_qaoa`, but the distributions overlap significantly across the 20 graph seeds, as indicated by the interquartile ranges (Q_1 to Q_3), so there is no practical difference in their ratios. At depth $p = 3$, the extrapolated transfer angles (`extrapolate_p3`) exhibit a lower mean approximation ratio than both `qaoa_opt` and `fixed_qaoa` across all graph sizes, indicating that linear-ramp extrapolation from $p = 2$ library angles loses accuracy when extended to $p = 3$.

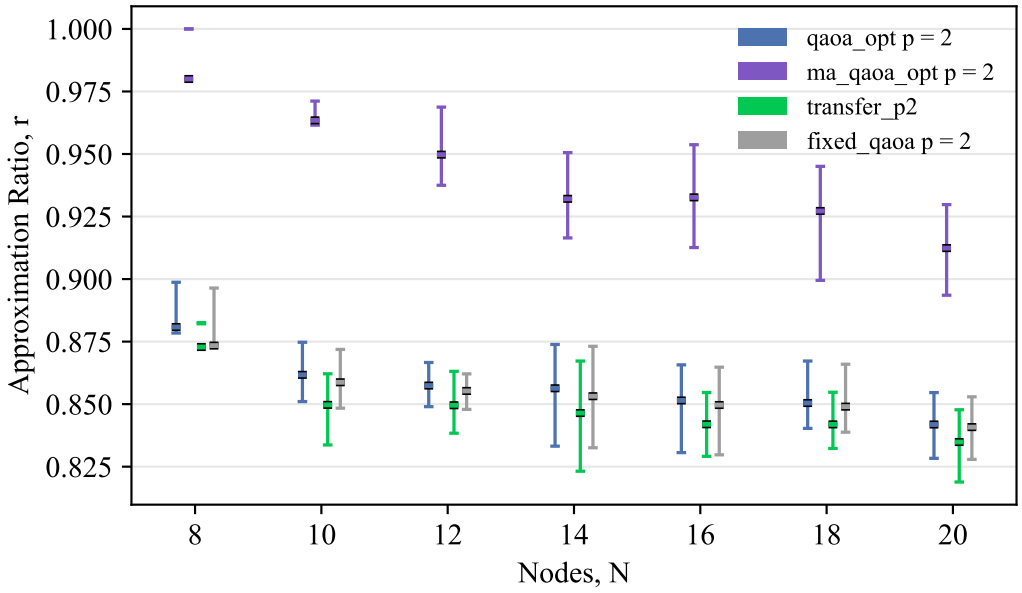

Figure 4.1: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at circuit depth $p = 2$. We compare subgraph transfer (`transfer_p2`), standard optimized QAOA (`qaoa_opt`), fixed-angle QAOA (`fixed_qaoa`), and fully optimized multi-angle QAOA (`ma_qaoa_opt`). Solid lines indicate the mean approximation ratio across 20 independent random 3-regular graph seeds, and whiskers indicate the first and third quartiles (Q_1 and Q_3). Symbols corresponding to the same N are slightly offset horizontally for clarity.

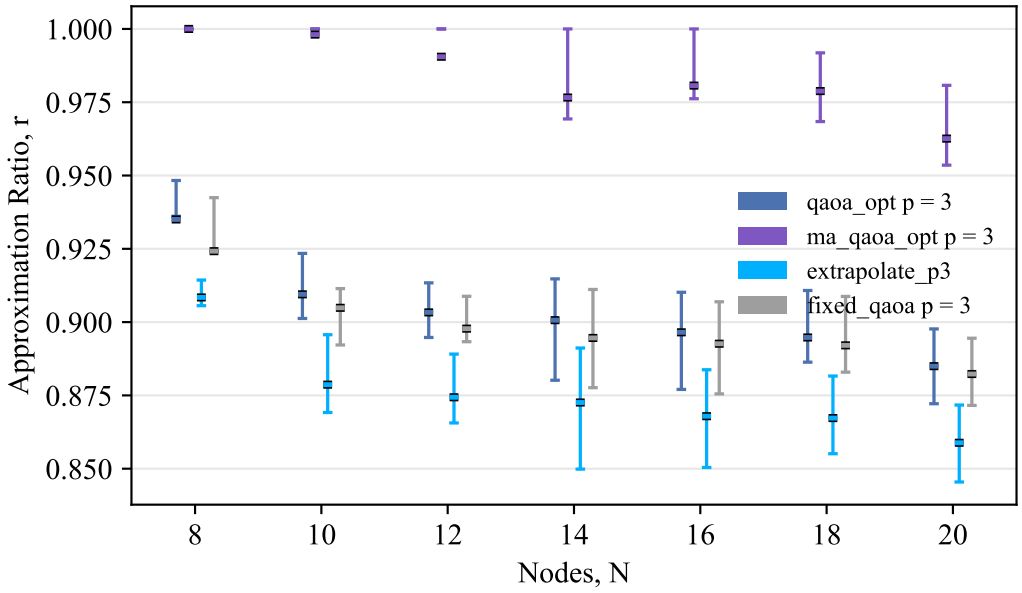

Figure 4.2: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at circuit depth $p = 3$. Subgraph transfer is evaluated using parameters extrapolated from the $p = 2$ library (`extrapolate_p3`, Section 3.3). Baselines include `qaoa_opt`, `fixed_qaoa`, and `ma_qaoa_opt`. Solid lines indicate the mean across 20 independent seeds, and whiskers show the Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

In standard QAOA, the variational state uses $2p$ independent parameters across the entire graph. By contrast, both `ma_qaoa_opt` and `transfer_p2` assign $p(|E| + |V|)$ local parameters ($5N$ total parameters for $d = 3$ regular graphs at $p = 2$). Despite utilizing the same number of variational parameters as `ma_qaoa_opt`, `transfer_p2` achieves mean approximation ratios comparable only to standard `qaoa_opt`.

Meanwhile, fixed-angle QAOA (`fixed_qaoa`), which uses $2p$ precomputed, unoptimized angles derived from a single tree-like subgraph, performs remarkably close to standard `qaoa_opt`. This robustness aligns with earlier findings (38,46), where it was demonstrated that tree-derived parameters do capture the dominant local structure of regular graphs.

During library construction (Section 3.2.1), each unique subgraph entry is optimized to maximize the expectation value $\langle H_e \rangle$ of its central edge in isolation. In that setting, the local p-depth neighborhood serves as the complete reverse causal cone of the edge (Section 2.6). In a full target graph G, however, every vertex and edge belongs to multiple overlapping causal cones simultaneously. Fully optimized ma-QAOA (`ma_qaoa_opt`) accounts for these overlaps by optimizing all $p(|E| + |V|)$ parameters jointly, allowing the angles to adjust to shared structural features across neighboring edges. The library transfer pipeline treats each local neighborhood independently and does not appear to adapt to this global context. Further, when transferring angles to a target graph, every vertex v receives multiple candidate mixer angles β_l from its incident edges. To resolve these conflicting assignments, the pipeline computes β_v as the average of the candidate angles (Section 3.2.2). This node-averaging step blends parameters derived from distinct local subgraphs, which can attenuate the specific optimization drive of individual edge environments and smooth out the contrast due to local environment.

4.2 Runtime Comparison

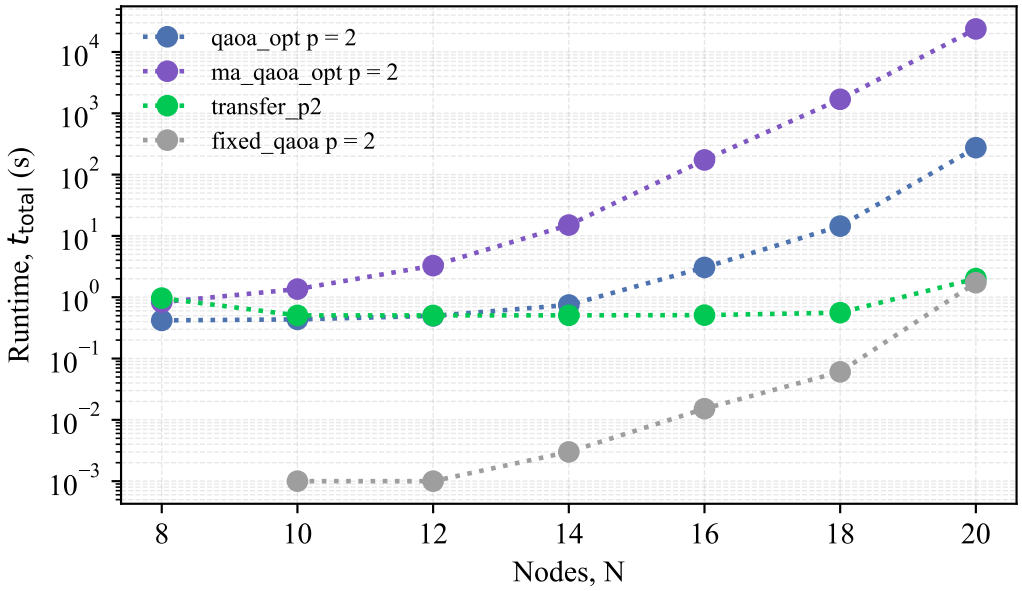

Figure 4.3: Total wall-clock runtime (in seconds) as a function of graph size N at circuit depth $p = 2$. We compare subgraph transfer (`transfer_p2`), standard optimized QAOA (`qaoa_opt`), fixed-angle QAOA (`fixed_qaoa`), and fully optimized multi-angle QAOA (`ma_qaoa_opt`). Runtimes are averaged over 20 independent random 3-regular graph seeds.

While the approximation ratio $r = \frac{\langle C \rangle}{C^*}$ measures the quality of the solution obtained from a parameterized state, we study the classical runtime required to simulate the entire algorithm to evaluate how computational effort scales with problem size. A central motivation for non-optimized strategies such as subgraph transfer (`transfer_p2`), parameter extrapolation (`extrapolate_p3`), and fixed-angle QAOA (`fixed_qaoa`) is to reduce simulation time by bypassing the substantial classical optimization overhead required by full-graph parameter training.

Figure 4.3 and Figure 4.4 illustrate the total wall-clock runtime (in seconds) as a function of graph size N for circuit depths $p = 2$ and $p = 3$ respectively.

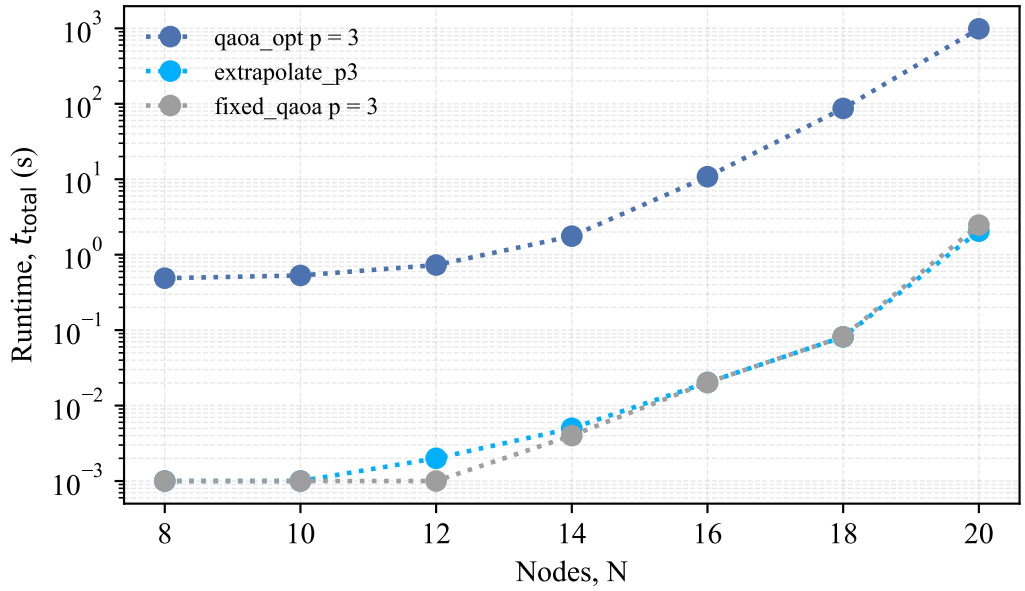

Figure 4.4: Total wall-clock runtime (in seconds) as a function of graph size N at circuit depth $p = 3$ for `extrapolate_p3`, `fixed_qaoa`, and `qaoa_opt`. Non-optimized methods requiring a single statevector evaluation scale identically with N, whereas `qaoa_opt` incurs 10 independent SLSQP restarts.[5]

Total wall-clock runtime can be divided into three main components: first, the time required for a single quantum circuit statevector evaluation; second, the total number of circuit evaluations requested by the classical optimizer; and third, the time required for classical pre- and post-processing.

Because every method simulates the same target graph G at depth p using Qiskit's `Statevector` backend, the time for a single circuit simulation is nearly identical across all methods for a given N. So, runtime differences are driven almost entirely by the other two components.

Both `fixed_qaoa` and `transfer_p2` (or `extrapolate_p3` at $p = 3$) require exactly one circuit statevector evaluation, as there is no parameter optimization step. For `fixed_qaoa`, classical processing time is negligible because precomputed tree angles are assigned directly across all layers. For `transfer_p2`, classical processing includes breadth-first search (BFS) subgraph extraction and library matching for every edge. At small N, where circuit simulation time is on the order of milliseconds, this BFS and library matching overhead makes `transfer_p2` slightly slower than `fixed_qaoa`. However, as N increases, circuit simulation time grows exponentially ($\mathcal{O}(2^N)$) and rapidly dominates classical processing time, causing the runtimes of `transfer_p2` and `fixed_qaoa` to converge.

Gradient-based optimization using SLSQP requires repeated statevector evaluations across multiple random restarts (8 restarts at $p = 2$ and 10 restarts at $p = 3$). Both standard QAOA (`qaoa_opt`) and ma-QAOA (`ma_qaoa_opt`) require multiple circuit simulation calls per restart, leading to substantially longer total runtimes before convergence.

Quantitatively, at depth $p = 2$ and $N = 20$, `ma_qaoa_opt` takes approximately two orders of magnitude longer to converge than `qaoa_opt`, which in turn takes approximately two orders of

[5] `ma_qaoa_opt` at $p = 3$ is omitted from this plot because it was run on separate hardware due to very long simulation time requirements, and hence can not be compared on the same scale.

magnitude longer than `transfer_p2` and `fixed_qaoa`. Overall, non-optimized methods achieve a $10^4\times$ reduction in total wall-clock time relative to full `ma_qaoa_opt` at $N = 20$.

At depth $p = 3$, `qaoa_opt` requires approximately three orders of magnitude longer than `fixed_qaoa` and `extrapolate_p3`. Meanwhile, `extrapolate_p3` maintains a runtime nearly identical to `fixed_qaoa` because linear-ramp slope calculation from $p = 2$ library entries adds negligible classical overhead[6].

When both accuracy and runtime are considered, `fixed_qaoa` proves advantageous over `transfer_p2` on 3-regular graphs, as it achieves a comparable or slightly higher approximation ratio while avoiding the classical graph-processing overhead associated with BFS extraction and library lookup (see Appendix D for behaviour at $N > 20$).

4.3 Semi-Transfer Strategy

In full subgraph parameter transfer (Section 3.2.2), both the edge cost angles γ_e and the node mixer angles β_v are transferred from the library without classical training on the target graph. However, as shown in Section 4.1, this combined transfer achieves approximation ratios comparable only to standard QAOA (`qaoa_opt`), despite using the same number of local parameters as fully optimized ma-QAOA (`ma_qaoa_opt`). To isolate the role of each parameter family and determine whether one set of gates possesses better transferability properties, we study two semi-transfer strategies at circuit depth $p = 2$, as described in Section 3.2.3: transferring β_{lib} while optimizing γ, and transferring γ_{lib} while optimizing β.

Both strategies reduce the number of optimizable free parameters relative to full ma-QAOA. For a d-regular graph with $d = 3$, the vertex and edge counts are $|V| = N$ and $|E| = 3\frac{N}{2}$, respectively. At depth $p = 2$, full `ma_qaoa_opt` optimizes $p(|E| + |V|) = 5N$ parameters (100 parameters at $N = 20$). Transferring β leaves $p|E| = 3N$ optimizable parameters (60 at $N = 20$), and transferring γ leaves $p|V| = 2N$ optimizable parameters (40 at $N = 20$). For comparison, standard `qaoa_opt` optimizes only $2p = 4$ global parameters.

[6] While a slight runtime difference exists between `transfer_p2` and `fixed_qaoa` at $p = 2$, this difference disappears between `extrapolate_p3` and `fixed_qaoa` at $p = 3$. This occurs because BFS subgraph extraction and library matching were performed once for `transfer_p2` and then cached. For `extrapolate_p3`, the cached matches are reused, leaving only the negligible classical computation required to evaluate the linear-ramp equations (Section 3.3.3).

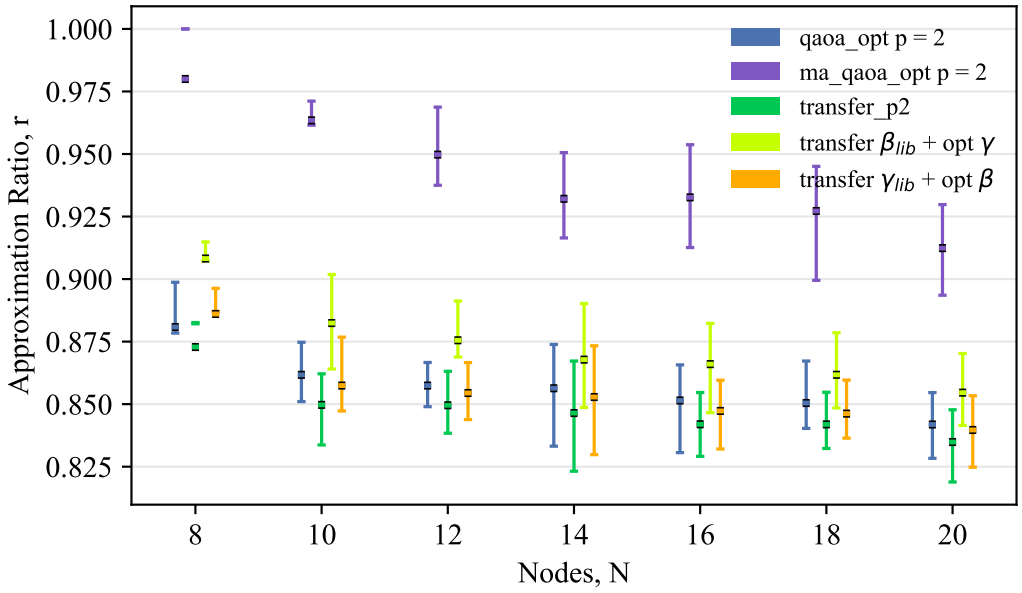

Figure 4.5: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at depth $p = 2$ for semi-transfer strategies. We compare `transfer` β_{lib} `+ opt` γ and `transfer` γ_{lib} `+ opt` β against baseline methods `qaoa_opt`, `ma_qaoa_opt`, and full `transfer_p2`. Solid lines indicate the mean across 20 independent random 3-regular graph seeds, and whiskers indicate the Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

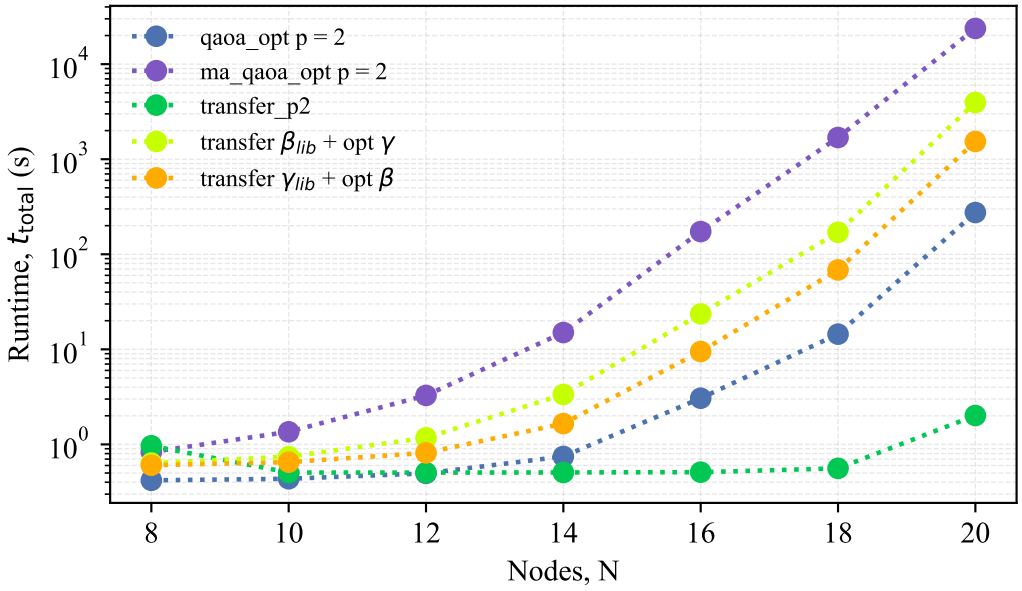

Figure 4.6: Total optimization wall-clock time (in seconds) as a function of graph size N at depth $p = 2$ for semi-transfer strategies alongside `qaoa_opt`, `ma_qaoa_opt`, and full `transfer_p2`. Runtimes are averaged over 20 independent seeds on a single thread.

Figure 4.5 presents the approximation ratio r as a function of N for these semi-transfer variants alongside baseline methods. Fully optimized `ma_qaoa_opt` maintains the highest mean approximation ratio across all graph sizes. Among the semi-transfer protocols, transferring β while optimizing γ (`transfer` β_{lib} `+ opt` γ) achieves a modest performance gain over standard QAOA. However, with increasing system size N, this gain diminishes. Furthermore, transferring γ while optimizing β (`transfer` γ_{lib} `+ opt` β) clusters closely with standard `qaoa_opt` and full `transfer_p2` in terms of performance.

Cost gates $e^{\left\{-i\left(\frac{\gamma_{\{e,l\}}}{2}\right)Z_iZ_j\right\}}$ directly generate entanglement across neighboring qubits and dictate the phase accumulation associated with specific edges. These edges are shared between multiple overlapping reverse causal cones (Section 2.6), and local cost parameters derived from isolated subgraphs cannot capture global phase interference. Optimizing γ_e directly on the host graph allows the optimizer to adjust these entangling phases jointly, slightly mitigating local misalignments across overlapping cones.

On the other hand, mixer gates $e^{\left\{-i\beta_{\{v,l\}}X_v\right\}}$ drive local single-qubit basis rotations. They act locally on individual vertices, and full optimization of this parameter family does not capture much of the global entanglement structure. Consequently, mixer parameters present an easier family of parameters to transfer compared to cost parameters for Max-Cut.

Figure 4.6 illustrates the corresponding classical optimization wall-clock runtimes. The runtime required by each method correlates directly with the number of optimizable parameters: `ma_qaoa_opt` ($5N$ parameters) incurs the highest wall-clock cost, followed by `transfer` β_{lib} `+ opt` γ ($3N$ parameters), `transfer` γ_{lib} `+ opt` β ($2N$ parameters), and `qaoa_opt` (4 parameters).

While `transfer` β_{lib} `+ opt` γ recovers a portion of the performance gap between standard QAOA and full ma-QAOA, it still requires a full classical optimization loop over $3N$ variables. At $N = 20$, its wall-clock runtime remains more than three orders of magnitude higher than non-optimized single-evaluation methods (`transfer_p2` and `fixed_qaoa`). Consequently, semi-transfer protocols do not provide a compelling practical alternative in their current form, as the modest gain in approximation ratio over standard or fixed-angle QAOA does not justify the substantial classical optimization overhead required at larger problem sizes.

4.4 Warm-Parameter Setting from Subgraph Transfer

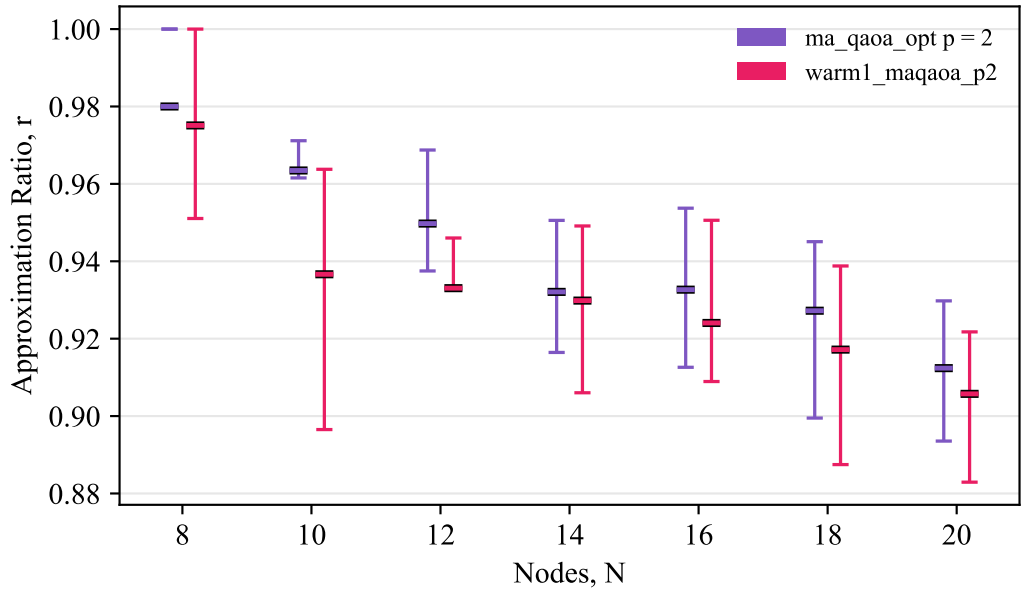

Figure 4.7: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at circuit depth $p = 2$ for cold-started full ma-QAOA (`ma_qaoa_opt`, 8 random restarts) and warm-started ma-QAOA initialized with transferred library parameters (1 restart). Solid lines represent mean values across 20 independent random 3-regular graph seeds, and whiskers indicate Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

In full ma-QAOA optimization (`ma_qaoa_opt`), denoted as cold-restarted ma-QAOA in this section, the non-convex parameter landscape has numerous local minima and barren plateaus, which requires multiple random restarts (8 at $p = 2$ and 10 at $p = 3$) to locate high-quality parameters. In this section, we study warm-started ma-QAOA optimization, in which the optimizer is initialized at the transferred (or extrapolated) parameter values and executed for only a single SLSQP optimizer restart (1 optimization run).

If transferred parameters are able to place the parameterized state close to a good local minimum, a single local optimization run can refine the angles without requiring exhaustive landscape exploration with multiple restarts across random initializations.

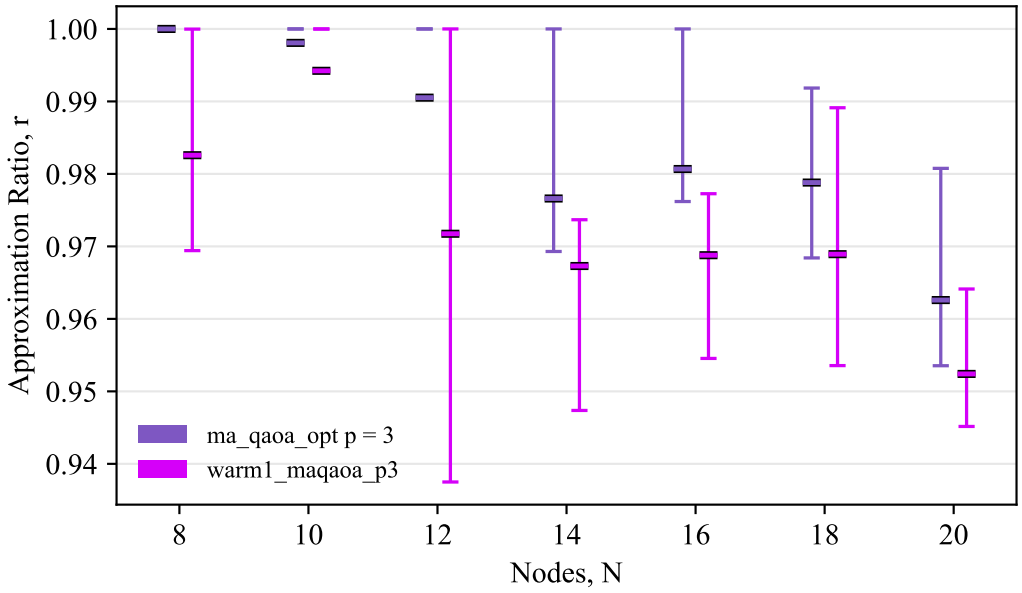

Figure 4.8: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at circuit depth $p = 3$ for cold-started ma-QAOA (`ma_qaoa_opt`, 10 random restarts) and warm-started ma-QAOA initialized with parameters extrapolated from the $p = 2$ library (Section 3.3.3, 1 restart). Solid lines show mean values across 20 independent seeds, and whiskers indicate Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

Figure 4.7 and Figure 4.8 compare the approximation ratios achieved by cold-started ma-QAOA and warm-started ma-QAOA at depths $p = 2$ and $p = 3$, respectively. At both depths, warm-started ma-QAOA achieves mean approximation ratios within $1 - 2\%$ of the cold-started baseline across all evaluated graph sizes. This demonstrates that the transferred (or extrapolated) parameters place the optimizer in a favorable region of the parameter space, allowing a single local optimization run to recover nearly the full performance of multi-start ma-QAOA.

As discussed in Section 4.2, one major factor contributing to total simulation time is total number of quantum circuit evaluation calls made by the classical optimizer, denoted by n_{fev}. Figure 4.9 shows n_{fev} as a function of N for cold-started and warm-started ma-QAOA at $p = 2$.

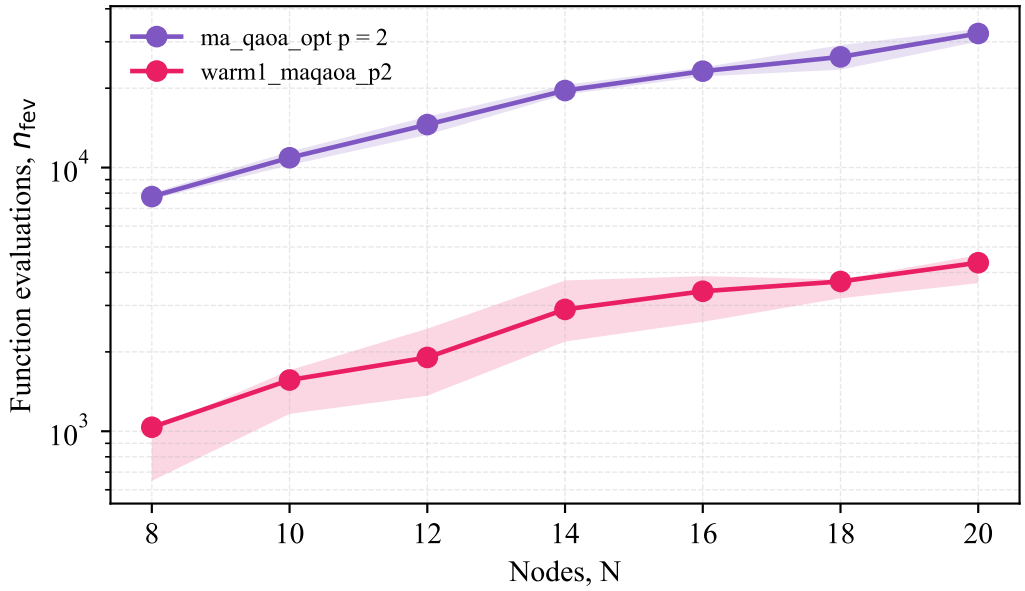

Figure 4.9: Total number of circuit function evaluations n_{fev} as a function of graph size N at circuit depth $p = 2$ for cold-started ma-QAOA (8 restarts) and warm-started ma-QAOA (1 restart). Counts are averaged over 20 independent seeds.

Because warm-started ma-QAOA runs only a single restart and starts closer to a local minimum, it requires approximately an order of magnitude fewer circuit evaluations (n_{fev}) to converge relative to cold-started ma-QAOA.

These results indicate that while subgraph transfer parameters alone do not match fully optimized ma-QAOA without local training (Section 4.1), they serve as an effective initialization heuristic. By replacing multiple random restarts with a single warm-started run, the classical optimization cost could be substantially reduced while preserving high approximation ratios.

4.5 Slope-Constrained ma-QAOA

In this work, we consider another parameter strategy, slope-constrained ma-QAOA (`ma_qaoa_slope_opt`), as introduced in Section 3.3.2. Previous works (5) have demonstrated the effectiveness of linear-ramp protocols for standard QAOA; here we extend this schedule locally to multi-angle circuits.

In standard ma-QAOA (`ma_qaoa_opt`), the number of free parameters is $p(|E| + |V|)$, growing linearly with both circuit depth p and graph size ($|E|$ and $|V|$). In slope-constrained ma-QAOA, we restrict the layer-wise parameters of each edge e and node v to linear ramps characterized by two slope variables, Δ_{γ_e} and Δ_{β_v} (Equation 30). As a result, the total number of optimizable parameters is $|E| + |V|$, which depends on graph size but remains independent of circuit depth p. For 3-regular graphs ($|E| = 3\frac{N}{2}$ and $|V| = N$), this imposes a constant $2.5N$ parameter count across all depths p, positioning this strategy between standard QAOA ($2p$ parameters) and full ma-QAOA ($5N$ parameters at $p = 2$, $7.5N$ at $p = 3$) in terms of total optimizable parameters, as summarized in Table 4.1.

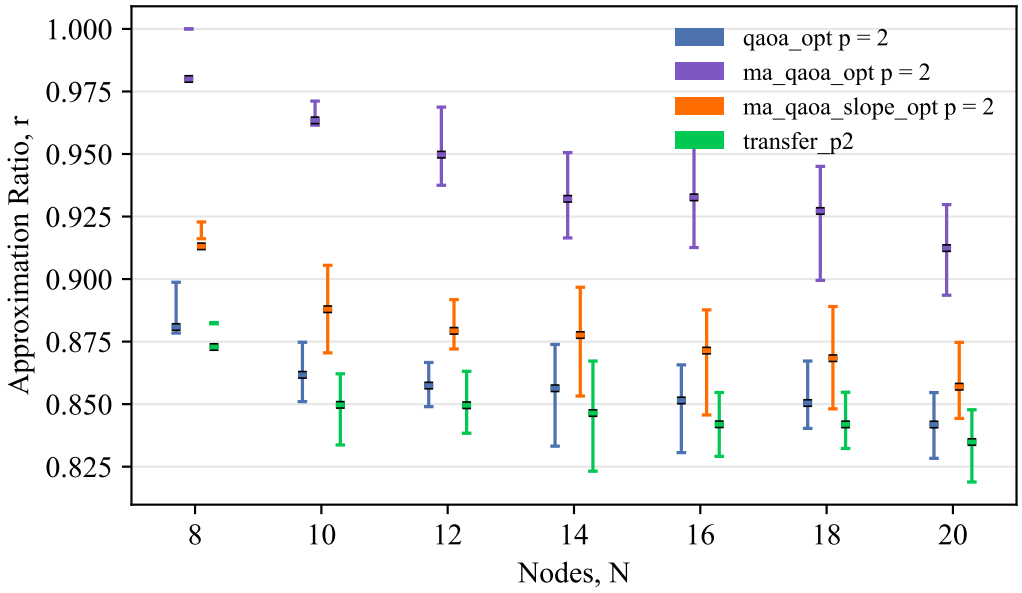

Figure 4.10: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at circuit depth $p = 2$ for slope-constrained ma-QAOA (`ma_qaoa_slope_opt`), standard QAOA (`qaoa_opt`), and fully optimized ma-QAOA (`ma_qaoa_opt`). Solid lines indicate mean values across 20 independent random 3-regular graph seeds, and whiskers show the Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

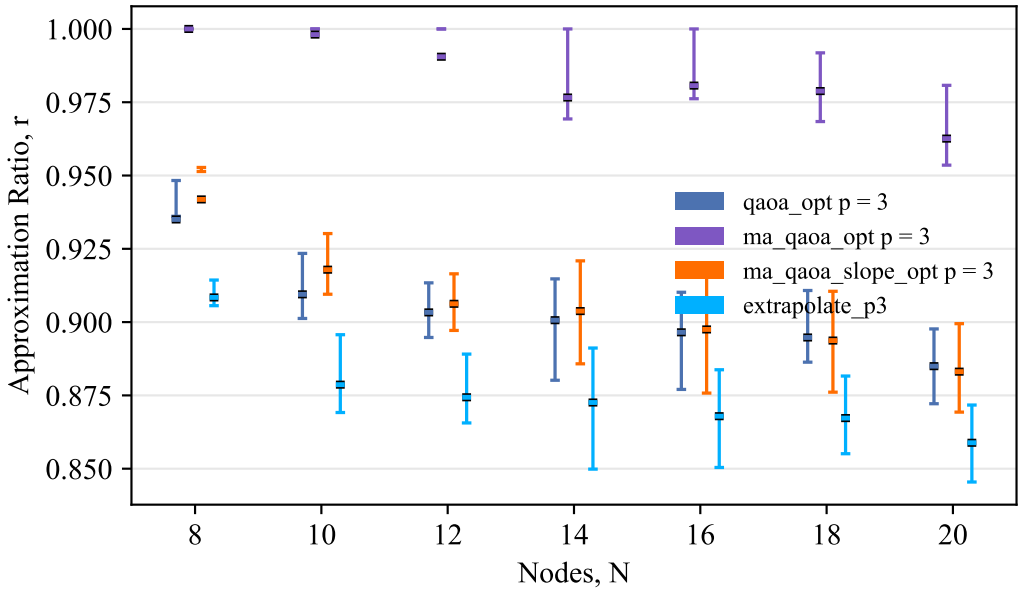

Figure 4.11: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N at circuit depth $p = 3$ for slope-constrained ma-QAOA (`ma_qaoa_slope_opt`), standard QAOA (`qaoa_opt`), and fully optimized ma-QAOA (`ma_qaoa_opt`). Solid lines represent means across 20 independent seeds, and whiskers indicate Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

Figure 4.10 and Figure 4.11 display the approximation ratios achieved by slope-constrained ma-QAOA alongside standard QAOA and full ma-QAOA for $p = 2$ and $p = 3$, respectively. At depth $p = 2$, `ma_qaoa_slope_opt` achieves a mean approximation ratio between standard `qaoa_opt` and full `ma_qaoa_opt`. However, as the graph size N increases, the performance gain of `ma_qaoa_slope_opt` over standard `qaoa_opt` gradually diminishes. At depth $p = 3$, this trend is more pronounced, and the approximation ratio curve of `ma_qaoa_slope_opt` sits much closer to standard `qaoa_opt` across the evaluated problem sizes.

Figure 4.12 and Figure 4.13 show the corresponding wall-clock runtimes of the complete simulation as a function of N for depths $p = 2$ and $p = 3$.

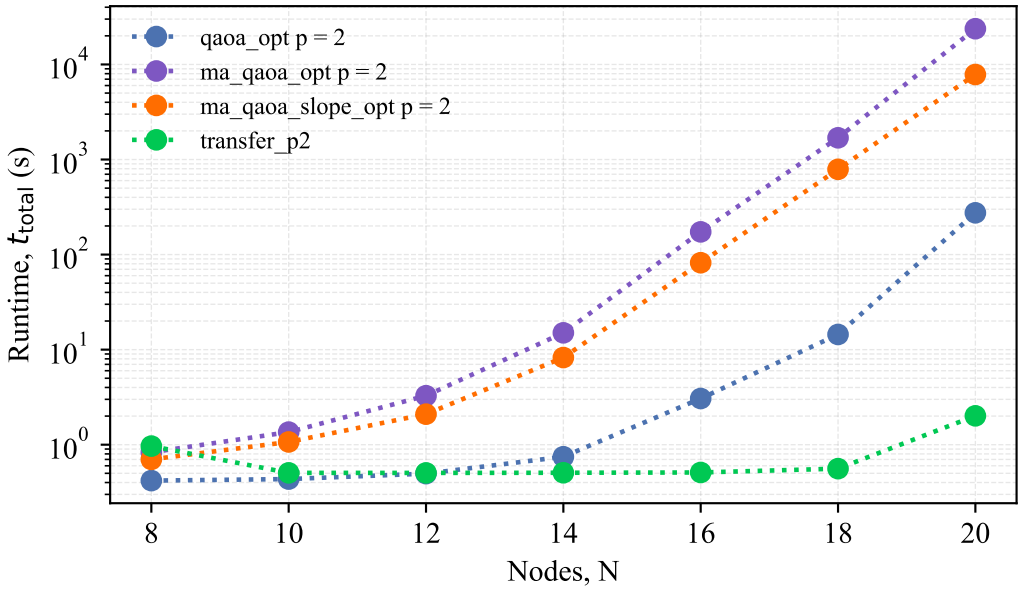

Figure 4.12: Total optimization wall-clock time (in seconds) as a function of graph size N at circuit depth $p = 2$ for slope-constrained ma-QAOA (`ma_qaoa_slope_opt`), standard QAOA (`qaoa_opt`), and full ma-QAOA (`ma_qaoa_opt`). Runtimes are averaged over 20 independent seeds on a single thread.

The runtime scales directly with the total number of optimizable parameters, with the runtime curve for `ma_qaoa_slope_opt` sitting between `qaoa_opt` and `ma_qaoa_opt`. The slope optimization strategy provides only a modest speedup when compared to full `ma_qaoa_opt`, whereas the drop in approximation ratio in this case is quite significant($5 - 7\%$ for p=2; $6 - 9\%$ for p=3).

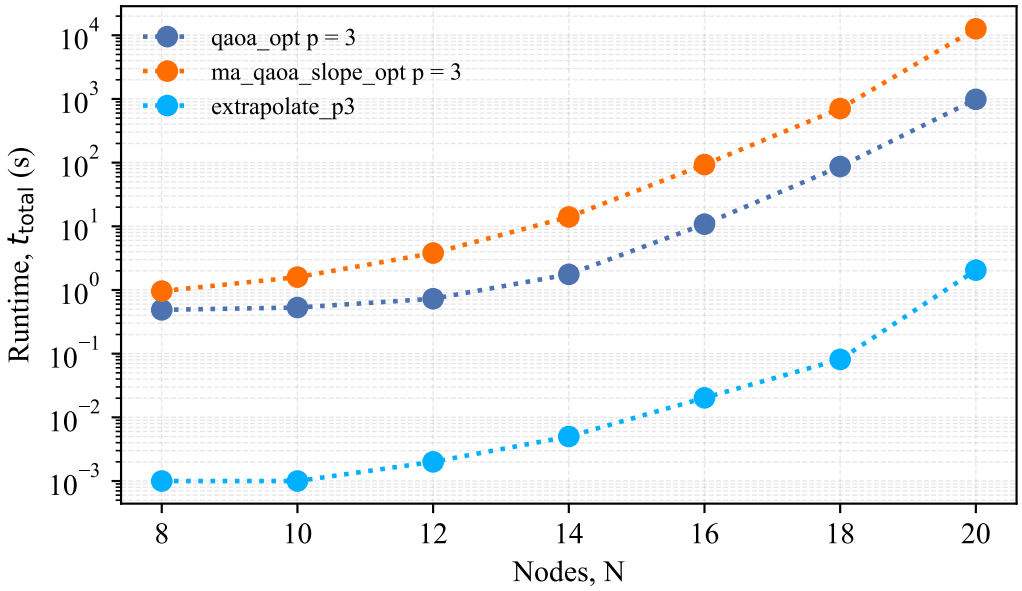

Figure 4.13: Total optimization wall-clock time (in seconds) as a function of graph size N at circuit depth $p = 3$ for slope-constrained ma-QAOA (`ma_qaoa_slope_opt`), standard QAOA (`qaoa_opt`), and full ma-QAOA (`ma_qaoa_opt`). Runtimes are averaged over 20 independent seeds on a single thread.

4.6 Chapter Summary

The results of this chapter paint a consistent picture across all parameter-setting strategies. Direct subgraph transfer, and its extrapolated extension at $p = 3$, do not outperform standard QAOA or fixed-angle QAOA on 3-regular graphs, despite assigning as many parameters as full ma-QAOA, because library angles are optimized in isolated causal cones and node averaging attenuates local parameter contrast. The semi-transfer simulations show that mixer angles transfer more reliably than cost angles, and the warm-start simulations show that transferred parameters nevertheless land close to a good local minimum: a single optimization run from this initialization recovers nearly the full ma-QAOA performance while cutting the number of circuit evaluations by an order of magnitude. Slope-constrained ma-QAOA reduces the parameter count to $|E| + |V|$ but loses its advantage over standard QAOA as depth and system size grow. In the next chapter, we summarize these findings, draw out their common explanation, and outline directions for future work.

Chapter 5

Conclusion and Outlook

5.1 Summary

In this thesis, we investigated parameter-setting strategies for multi-angle QAOA (ma-QAOA) applied to the Max-Cut problem on 3-regular graphs. While ma-QAOA enhances parameterized state expressivity and yields higher approximation ratios than standard QAOA, it also introduces a combinatorial explosion of variational parameters ($p(|E| + |V|)$). We reviewed how large parameter dimension severely complicates classical optimization and the evolution of the parameterized state through quantum gates. The central objective of this work was to evaluate whether parameter transfer and regularization methods can bypass or accelerate full-graph optimization while retaining the performance advantages of ma-QAOA.

We explored two primary parameter-setting frameworks and several hybrid variants:

1. Subgraph Parameter Transfer: Motivated by the principle of light-cone locality (Section 2.6), we constructed a complete pre-optimized library of unique p-depth subgraphs ($p \in \{1, 2\}$). Target graph edges were matched to library entries via Weisfeiler-Lehman hashing and rooted isomorphism (Appendix C), transferring precomputed cost parameters γ_e directly and deriving node mixer parameters β_v through averaging over all incident edges of each node (Section 3.2.2). Despite assigning $p(|E| + |V|)$ parameters without target-graph training, direct transfer (`transfer_p2`) achieved mean approximation ratios comparable only to standard single-angle QAOA (`qaoa_opt`). This performance limitation arises because library subgraphs optimize edges in isolated causal cones and fail to account for overlapping causal environments present in target graphs.
2. Semi-Transfer Protocols: To isolate the transferability of individual parameter families (Section 3.2.3), we considered two hybrid protocols at depth $p = 2$. Transferring node mixer parameters β_v while optimizing edge cost parameters γ_e ($p|E| = 3N$ parameters) achieved a noticeable performance gain over standard QAOA at smaller system sizes. However, transferring γ_e while optimizing β_v ($p|V| = 2N$ parameters) collapsed back to the standard QAOA baseline. Both parameter families play different roles in how they carry forward information about the target graph. Cost parameters γ_e control entangling phases across overlapping causal cones, carrying a "global" picture of the host target graph; on the other hand, mixer parameters carry a "local" picture and are less sensitive to global cone interference.
3. Warm-Started Optimization: We evaluated library-transferred (at $p = 2$) and linear-ramp-extrapolated (at $p = 3$) parameters as initializations for full ma-QAOA optimization using a single

SLSQP restart (Section 3.2.3). Warm-started ma-QAOA achieved approximation ratios within 1–2% of multi-start cold-initialized ma-QAOA while reducing the required circuit function evaluations (n_{fev}) by approximately an order of magnitude. This shows that the transferred library parameters land inside or close to favorable basins of attraction, reducing the need for exhaustive multi-restart landscape searching.

4. Slope-Constrained ma-QAOA: To reduce optimization dimensionality, we extended linear-ramp schedules locally to multi-angle circuits, constraining parameters to two slope variables, one per edge (Δ_{γ_e}) and one per node (Δ_{β_v}) (Section 3.3). This fixed the total parameter count to $|E| + |V|$ ($2.5N$ for $d = 3$), independent of depth p. While slope-constrained ma-QAOA (`ma_qaoa_slope_opt`) outperformed standard QAOA at $p = 2$, its performance advantage almost diminished at $p = 3$ and larger N, approaching standard QAOA as the linear constraint restricted parameterized state expressivity.

Across all non-optimized strategies, fixed-angle QAOA (`fixed_qaoa`), which uses $2p$ unoptimized angles derived from a single infinite-tree subgraph, proved to be a strong practical choice on 3-regular graphs. It matched or exceeded direct subgraph transfer in approximation ratio while avoiding subgraph library creation, library optimization, classical graph extraction, and lookup overhead.

In summary, our findings demonstrate that the rigid parameter constraints considered in this work, whether direct subgraph transfer, or linear ramps, yield performance levels comparable to standard QAOA. To fully realize the superior approximation ratios of ma-QAOA, the classical optimizer may require full freedom to optimize all $p(|E| + |V|)$ angles jointly across overlapping causal cones, though warm-started initialization offers an effective route to accelerate that convergence.

5.2 Outlook

Building upon the findings of this thesis, several promising research directions can be pursued to further understand and improve parameter-setting strategies for multi-angle QAOA.

1. Capturing Overlapping Causal Context: Subgraph library transfer evaluates each local p-depth neighborhood in isolation, neglecting the overlapping reverse causal cones present in full target graphs (Section 4.1). Future work could extend the library construction to incorporate this neighboring structural context as well. For instance, by optimizing library parameters over extended subgraphs that include second-nearest edge environments or by developing some context-aware parameter update rules beyond simple transfer.
2. Lie-Algebraic Analysis of Constrained Parameterized state: A formal Lie-algebraic characterization of constrained ma-QAOA parameterized state would provide valuable theoretical insight. By analyzing the Dynamical Lie Algebra (DLA) generated by restricted parameter schedules, we could formally quantify the expressivity and Hilbert space reachability of these constrained states relative to full $\mathfrak{g}_{\text{free}}$, rigorously explaining where performance bottlenecks originate (Section B), and justify why they tend to fall back to the approximation ratios of standard QAOA.
3. Large-Scale Benchmarking: As problem size scales, the classical optimization cost of full ma-QAOA becomes entirely prohibitive, making non-optimized heuristics the only feasible alternative. In Appendix D, we conducted preliminary evaluations comparing `transfer_p2` (and `extrapolate_p3`) against `fixed_qaoa` using Matrix Product State (MPS) simulations for graph sizes up to $N = 80$. Across these larger instances, approximation ratios remain close (within 0.5% at $p = 2$ and 2-4% at $p = 3$). However, runtime measurements indicate that transfer routines become computationally competitive with fixed-angle evaluations, in contrast to the earlier results. This work could be extended by investigating the tensor network simulation dynamics for these larger graph sizes.

4. Irregular and Inhomogeneous Graph Topologies: Finally, the parameter transfer pipeline should be evaluated on broader graph classes, such as irregular graphs with varying degree distributions, or weighted Max-Cut instances. On homogeneous 3-regular graphs, all edges share identical local degree constraints, which heavily favors uniform tree-derived fixed parameters ((26)). On irregular or weighted graphs, where local edge environments exhibit significant structural variance, fixed angles might fail to adapt to local variations. Testing transfer pipelines on inhomogeneous graphs will help identify the exact structural regimes where these constrained multi-angle transfer could provide some decisive advantage.

Appendices

Appendix A

Derivation of the Reverse Causal Cone Bound

This appendix gives the algebraic details behind the light-cone argument of Section 2.6. The goal is to show that the reverse causal cone of an edge operator is contained in the p-depth subgraph induced around that edge.

1.1 Setup and edge decomposition

The expectation value of the cost Hamiltonian on the p-layer QAOA state is

$$\langle C \rangle = \langle + \,|^{\otimes n}\, U_1^\dagger \cdots U_p^\dagger H_C U_p \cdots U_1 \,|\, + \rangle^{\otimes n}, \tag{38}$$

where $U_\ell = U_M(\beta_\ell) U_C(\gamma_\ell)$ is the unitary of layer ℓ. The cost Hamiltonian decomposes into commuting edge terms,

$$H_C = \sum_{e \in E} C_e, \quad C_e = \frac{1}{2}\big(I - Z_i Z_j\big) \quad \text{for } e = (i,j), \tag{39}$$

so by linearity $\langle C \rangle = \sum_e \langle C_e \rangle$. The identity part of C_e contributes a constant and has empty support; the cone structure is therefore determined by the central operator $Z_i Z_j$.

Both unitaries factor into local terms. The cost unitary is

$$U_C(\gamma) = \prod_{e \in E} e^{-i\gamma C_e} \equiv \prod_{(u,v) \in E} e^{i\gamma \frac{Z_u Z_v}{2}}, \tag{40}$$

where the equivalence holds up to a global phase, since $e^{-i\gamma C_{uv}} = e^{-i\frac{\gamma}{2}} e^{i\gamma Z_u \frac{Z_v}{2}}$. The mixer unitary is

$$U_M(\beta) = \prod_{v \in V} e^{-i\beta X_v}. \tag{41}$$

1.2 Backward Propagation of central edge operator

Let us begin by fixing an edge $e = (i,j)$ and define the top-layer observable $O_e^{(p)} = Z_i Z_j$. Propagating it backward through layer ℓ gives

$$O_e^{(\ell-1)} = U_\ell^\dagger O_e^{(\ell)} U_\ell = U_C^\dagger(\gamma_\ell) U_M^\dagger(\beta_\ell) O_e^{(\ell)} U_M(\beta_\ell) U_C(\gamma_\ell). \tag{42}$$

We track how the vertex support $S_\ell := \mathrm{supp}\big(O_e^{(\ell)}\big)$ changes under the two conjugations(Mixer and Cost unitary).

1.3 Mixer action

A single-qubit mixer rotation acts on Pauli operators as

$$e^{i\beta X_v} Z_v e^{-i\beta X_v} = Z_v \cos(2\beta) + Y_v \sin(2\beta), \quad e^{i\beta X_v} Y_v e^{-i\beta X_v} = Y_v \cos(2\beta) - Z_v \sin(2\beta), \tag{43}$$

which follow from the Hadamard lemma $e^A B e^{-A} = B + [A,B] + \frac{1}{2}[A,[A,B]] + \cdots$ together with the Pauli commutation relations. Applying this to the central term,

$$U_M^\dagger(\beta) Z_i Z_j U_M(\beta) = (Z_i \cos(2\beta) + Y_i \sin(2\beta))\big(Z_j \cos(2\beta) + Y_j \sin(2\beta)\big). \tag{44}$$

All resulting Pauli strings still act only on $\{i,j\}$. Since the mixer is a product of single-qubit gates, it does not enlarge the support and S_ℓ is unchanged by U_M.

1.4 Cost action

For a cost edge (u,v), conjugation by $U_{uv}^C(\gamma) = e^{i\gamma \frac{Z_u Z_v}{2}}$ gives

$$\big(U_{uv}^C\big)^\dagger X_u U_{uv}^C = X_u \cos\gamma + Y_u Z_v \sin\gamma, \quad \big(U_{uv}^C\big)^\dagger Y_u U_{uv}^C = Y_u \cos\gamma - X_u Z_v \sin\gamma, \tag{45}$$

while Z_u and Z_v commute with U_{uv}^C and are unchanged. Based on this we have three cases that follow:

- If (u,v) is disjoint from the support, the gate commutes with every Pauli string in the operator and cancels against its adjoint.
- If both endpoints are inside the support, the operator is modified but no new vertex appears.
- If exactly one endpoint is inside the support and the Pauli on it is X or Y, the sine terms in Equation 45 attach a Z on the outside vertex, adding it to the support.

Hence a cost layer can add only vertices adjacent to the current support, one at max.

1.5 Support growth and subgraph containment

To see how the support grows, let S_ℓ be the set of qubits (or vertices) in our support at layer ℓ. Initially, before any backward steps are taken, the support consists of only the two vertices of our starting edge: $S_p = \{i,j\}$. As we move backward through each layer ℓ, the support can only expand by including the direct neighbors of the vertices currently in S_ℓ. Let $N(S_\ell)$ be the set of all vertices in the graph that are directly connected by an edge to any vertex currently in S_ℓ. Since the support can grow by at most one neighboring vertex at each layer, the support at the next step, $S_{\ell-1}$, is always contained within the current support and its direct neighbors:

$$S_{\ell-1} \subseteq S_\ell \cup N(S_\ell). \tag{46}$$

If we repeat this step-by-step expansion backward through all p layers of the circuit, the final support of the operator, S_0, can only contain vertices that are at most p steps away from our starting vertices i and j:

$$S_0 \subseteq B_p(\{i,j\}) = \{v \in V : \mathrm{dist}_G(v,\{i,j\}) \le p\}, \tag{47}$$

where $B_p(\{i,j\})$ represents the neighborhood of all vertices within a graph distance p from our starting edge (i,j). Let us define $G_e^{(p)} = \big(V_e^{(p)}, E_e^{(p)}\big)$ as the local p-depth subgraph of the main graph, which contains all vertices within distance p of the edge $e = (i,j)$ and all edges connecting

them. Because our final operator support S_0 and all the gates that do not cancel are restricted to this local neighborhood, the actual reverse causal cone graph, $R_e^{(p)} = (V_0, E_0)$, is always a subgraph of this local p-depth neighborhood:

$$R_e^{(p)} \subseteq G_e^{(p)}. \tag{48}$$

This containment relation is exact. In the worst-case scenario (such as a tree-like graph with no cycles), the causal cone $R_e^{(p)}$ is exactly equal to the local subgraph $G_e^{(p)}$. On graphs with cycles, some terms may cancel out, making the actual causal cone even smaller than $G_e^{(p)}$.

1.6 Evaluating the local expectation value

Because the final evolved operator $O_e^{(0)}$ acts only on the qubits in our final support S_0, it acts as the identity operator I on all other qubits in the system. This allows us to separate the qubits of the system into two groups. The qubits outside our support S_0 are completely untouched by the operator. Since the identity operator has an expectation value of exactly 1 on the $|+\rangle$ state ($\langle + \mid I \mid + \rangle = 1$), these untouched qubits simply factor out of our calculation and can be ignored. The expected value of our edge operator C_e therefore depends only on the qubits inside the local p-depth subgraph $G_e^{(p)}$ and on the parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ applied within this subgraph:

$$\langle C_e \rangle = F_e\Big(G_e^{(p)}; \boldsymbol{\gamma}, \boldsymbol{\beta}\Big), \tag{49}$$

where F_e is a local function that evaluates the expectation value of the edge e using only the structure of the local subgraph $G_e^{(p)}$. Finally, summing this local expectation over all edges in the graph, we obtain the exact total expected cut value:

$$\langle C \rangle = \sum_{e \in E} F_e\Big(G_e^{(p)}; \boldsymbol{\gamma}, \boldsymbol{\beta}\Big). \tag{50}$$

Appendix B

Dynamical Lie Algebra and Barren Plateaus

This appendix presents the mathematical framework of Lie algebras used to analyze the trainability of standard QAOA and multi-angle ma-QAOA. We define the dynamical Lie algebra, explain its relationship to the barren-plateau phenomenon, and show why ma-QAOA is exponentially more vulnerable to vanishing gradients than standard QAOA.

2.1 Foundational Definitions

In a quantum circuit, the operations we can perform on a quantum state are determined by the circuit's gates. Each gate in our circuit is generated by a fixed Hermitian operator H_j, which evolves the state under the unitary $e^{-i\theta H_j}$. To understand the full range of states and dynamics the circuit can access, we study the algebra generated by these operators.

Let $\mathcal{G} = \{H_j\}$ be the set of **generators** applied in the circuit. In standard QAOA, the generator set consists of only the two global Hamiltonians:

$$\mathcal{G}_{\text{std}} = \{H_C, H_M\}, \tag{51}$$

where H_C is the cost Hamiltonian (Equation 14) and H_M is the mixer Hamiltonian (Equation 16). In ma-QAOA, this generator set is enlarged to include every individual local edge and node operator:

$$\mathcal{G}_{\text{ma}} = \{Z_u Z_v\}_{(u,v)\in E} \cup \{X_v\}_{v\in V}, \tag{52}$$

amounting to a total of $|E| + |V|$ independent generators. (Note: The physical Max-Cut cost operator for an edge is $\frac{I - Z_u Z_v}{2}$, but the identity I commutes with all operators and is omitted here as it does not affect the generated Lie algebra).

To capture the physical dynamics, we define the **Dynamical Lie Algebra** (DLA), denoted $\mathfrak{g}$, as the real linear span of the generators under the commutator bracket $[A, B] = AB - BA$. Specifically, we include the imaginary factor i because the unitaries are exponentials of anti-Hermitian operators (iH_j):

$$\mathfrak{g} = \text{span}_{\text{Lie}}\big(\{iH_j : H_j \in \mathcal{G}\}\big). \tag{53}$$

The DLA contains all generators, their real linear combinations, and all their nested commutators (for example, $[A, [B, C]]$).

The DLA is of fundamental importance because it determines the **Lie group** $e^{\mathfrak{g}}$. This group contains every unitary operator the parameterized circuit can possibly implement, regardless of

depth. A larger DLA dimension indicates a more expressive circuit capable of preparing a wider variety of quantum states, but it also directly controls the classical difficulty of finding those states.

2.2 The Barren-Plateau Theorem

A **barren plateau** is a regime where the cost landscape becomes exponentially flat as the system size scales (27). More precisely, the average gradient of the expectation value $\langle C \rangle$ is zero, and its variance vanishes exponentially with the number of qubits n. If the variance is exponentially small, resolution of the gradient requires an exponentially large number of measurement shots, causing classical optimization to stall.

Using the DLA framework, Kazi et al. (39) computed the exact variance of the cost function gradient for the ma-QAOA parametrised state. For a graph with $|E|$ edges and Hilbert space dimension $D = 2^n$, the variance of the partial derivative of the cost function C with respect to any variational parameter θ_i is given by:

$$\mathrm{Var}[\partial_i C] = \frac{4D^2\ |E|}{(D^2-4)(D+2)} \approx \frac{4\ |E|}{2^n}. \tag{54}$$

Because the dimension of the ma-QAOA DLA is $\dim(\mathfrak{g}) = O(4^n)$, this variance scales as $\mathcal{O}\big(\frac{1}{2^n}\big)$, which corresponds to $O\Big(\frac{1}{\sqrt{\dim(\mathfrak{g})}}\Big)$. This confirms that if the DLA dimension grows exponentially with the qubit count, the variance of the gradient is guaranteed to vanish exponentially, triggering a severe barren plateau. However, if the DLA dimension is restricted by symmetry to grow only polynomially with the system size ($\dim(\mathfrak{g}) = \mathcal{O}(\mathrm{poly}(n))$), the gradient remains resolvable and classical optimization can remain tractable.

2.3 Standard vs. Multi-Angle Lie Algebra Hierarchy

To understand why ma-QAOA is exceptionally difficult to train classically, we look at the DLA of both algorithms on Max-Cut.

The Lie algebra of standard QAOA, denoted $\mathfrak{g}_{\mathrm{std}}$, is highly restricted by the uniform parameter constraints, meaning all edges share the same angle γ and all nodes share β. For structured graphs, $\mathfrak{g}_{\mathrm{std}}$ can be low-dimensional, keeping standard QAOA relatively close to a navigable, trainable trajectory.

In contrast, by giving every edge and node its own parameter, ma-QAOA enlarges the generator set, yielding a much larger DLA. Kazi et al. (39) also showed that for "archetypal" connected graphs (which include the random 3-regular graphs studied in this thesis, and most graph families of practical interest), the ma-QAOA Lie algebra is isomorphic to:

$$\mathfrak{g}_{\mathrm{free}} = \mathfrak{su}\big(2^{n-1}\big) \oplus \mathfrak{su}\big(2^{n-1}\big), \tag{55}$$

where $\oplus$ represents the direct sum of Lie algebras.

Here, $\mathfrak{su}(d)$ represents the special unitary Lie algebra of dimension $d^2 - 1$ (the set of traceless $d \times d$ anti-Hermitian matrices). Physically, $\mathfrak{su}\big(2^{n-1}\big) \oplus \mathfrak{su}\big(2^{n-1}\big)$ represents the full, unconstrained set of operations restricted only by the bit-flip ($X^{\otimes n}$) parity symmetry (the even and odd parity sectors of the n-qubit space). The dimension of this algebra is:

$$\dim(\mathfrak{g}_{\mathrm{free}}) = 2\Big(\big(2^{n-1}\big)^2 - 1\Big) = 2^{2n-1} - 2 = O(4^n). \tag{56}$$

Because standard QAOA is a constrained case of ma-QAOA, their DLAs satisfy the inclusion relation:

$$\mathfrak{g}_{\text{std}} \subseteq \mathfrak{g}_{\text{free}}. \tag{57}$$

This algebraic hierarchy explains the severe optimization bottleneck of ma-QAOA. By moving from $\mathfrak{g}_{\text{std}}$ to $\mathfrak{g}_{\text{free}}$, ma-QAOA gains massive expressivity but is mathematically highly prone to entering the barren-plateau regime because its DLA dimension $\dim(\mathfrak{g})$ is exponentially large.

This trainability barrier is the physical reason we cannot rely on classical optimizers to train ma-QAOA directly on large graphs. This opens up a need to use cheap, non-trivial parameter-setting heuristics to bypass this high-dimensional barren plateau.

Appendix C

Graph Isomorphism and the Weisfeiler-Lehman Hash

This appendix introduces the two graph-comparison tools used in our subgraph-library construction: graph isomorphism and the Weisfeiler-Lehman (WL) hash. Together, these tools allow us to determine efficiently whether a newly generated subgraph is structurally identical to an entry already stored in the library, preventing redundant optimization of duplicate subgraphs.

3.1 Graph Isomorphism

Two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be **isomorphic**, written $G_1 \cong G_2$, if there exists a bijection $f : V_1 \to V_2$ that preserves adjacency. That is, for every pair of vertices $u, v \in V_1$:

$$\{u, v\} \in E_1 \quad \text{if and only if} \quad \{f(u), f(v)\} \in E_2. \tag{58}$$

Intuitively, two graphs are isomorphic if one can be obtained from the other simply by relabelling its vertices. The underlying connectivity structure remains identical and only the names assigned to the vertices differ.

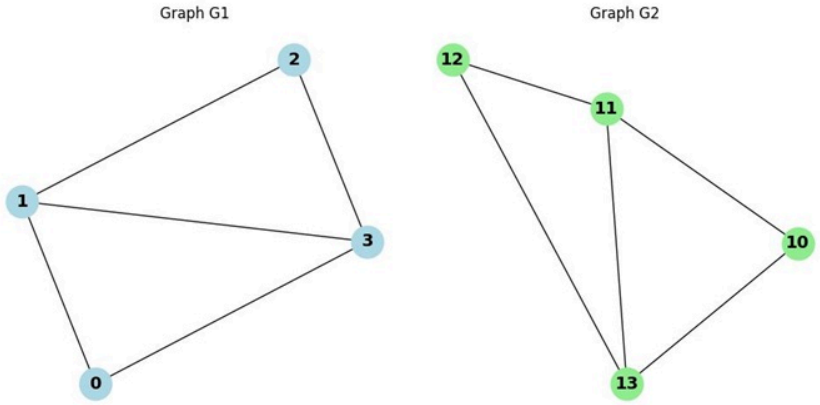

Figure C.1: Two four-vertex graphs with different vertex labels. Although drawn differently, G_1 and G_2 are isomorphic: the relabelling mapping $f(0) = 12$, $f(1) = 11$, $f(2) = 10$, and $f(3) = 13$ maps every edge of G_1 directly onto an edge of G_2 (6).

To illustrate this definition, consider the two graphs shown in Figure C.1. Graph G_1 is defined by the vertex set $V_1 = \{0, 1, 2, 3\}$ and edge set $E_1 = \{\{0, 1\}, \{0, 3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$. Graph G_2 has the vertex set $V_2 = \{10, 11, 12, 13\}$ and edge set $E_2 = \{\{10, 11\}, \{10, 13\}, \{11, 12\}, \{11, 13\}, \{12, 13\}\}$. Placed side by side, their drawings and labels look different. However, both graphs consist of a four-vertex cycle closed by a single diagonal chord, and the mapping

$$f(0) = 12, \quad f(1) = 11, \quad f(2) = 10, \quad f(3) = 13 \tag{59}$$

is a valid isomorphism. Checking each edge of G_1 under this mapping confirms that every edge in E_1 maps to a valid edge in E_2: $\{0, 1\} \to \{12, 11\}$, $\{0, 3\} \to \{12, 13\}$, $\{1, 2\} \to \{11, 10\}$, $\{1, 3\} \to \{11, 13\}$, and $\{2, 3\} \to \{10, 13\}$.

In our subgraph library, we want to store only unique, non-isomorphic graphs. For the pair in Figure C.1, the isomorphism check would return `True`, and the duplicate graph would be discarded. A different four-vertex structure, such as a simple cycle or a star graph, would fail the isomorphism test and be stored as a new, unique library entry.

The exact computational complexity of the graph isomorphism problem is still an open question in computer science. It belongs to the NP complexity class but is not known to be NP-complete or NP-hard. It was shown by Babai (47) to be solvable in quasi-polynomial time. For the small subgraphs encountered in our library (typically of size $|V| \leq 30$ at depth $p = 3$), exact isomorphism checking is highly tractable. However, because checking every candidate against thousands of existing library entries is computationally expensive, we use the WL hash as a rapid preliminary filter before running the full isomorphism check.

3.2 The Weisfeiler-Lehman Hash

The Weisfeiler-Lehman (WL) hash is a fast, heuristic graph fingerprint that assigns a compact label (a hash) to a graph based on its local connectivity structure. It is derived from the Weisfeiler-Lehman colour refinement algorithm (48). The key property of the WL hash is that if two graphs have different hashes, they are guaranteed to be non-isomorphic. If they share the same hash, they are highly likely, but not guaranteed, to be isomorphic. This makes the WL hash an excellent necessary condition for isomorphism, allowing us to quickly rule out non-matching pairs in constant time.

To compute the WL hash for our library, we apply a rooted colour refinement process. We assign one initial colour label to the "root" nodes of our local p-depth subgraph (the endpoints of the central edge), and a different colour label to all other vertices. This initial differentiation ensures that the final WL hash is sensitive not only to the general connectivity of the subgraph but also to the specific role each vertex plays relative to the central edge. In our library construction pipeline, the WL hash and the exact isomorphism test are used sequentially. When a new candidate subgraph is generated, we compute its rooted WL hash and compare it against the hashes of the subgraphs already stored in the library. If no existing entry shares this hash, the candidate is guaranteed to be unique and is immediately added to the library. If a hash collision occurs, we run an exact rooted isomorphism test (using NetworkX's isomorphism solver) against only the entries that share that specific hash. The candidate is added to the library only if it is found to be non-isomorphic to those entries; otherwise, it is discarded as a duplicate. This two-stage approach reduces the number of expensive isomorphism checks during library construction, accelerating the pipeline.

Appendix D

Transfer Strategy at Larger Graph Sizes

In Chapter 4, all comparisons were conducted up to $N = 20$ using Qiskit's exact `Statevector` simulator. In this section, we extend the comparison to larger graph instances, for non-optimized single-evaluation methods: subgraph transfer (`transfer_p2`), parameter extrapolation (`extrapolate_p3`), and fixed-angle QAOA (`fixed_qaoa`). We have considered same 3- regular graphs for $N \in \{30, 36, 40, 50, 60, 70, 80\}$.

At these graph sizes, exact statevector simulation required large amount of RAM to store the double-precision complex amplitudes. So, for these simulation we use Matrix Product State (MPS) tensor-network backend via Qiskit Aer's `matrix_product_state` simulator, with a maximum bond dimension of $\chi = 512$.

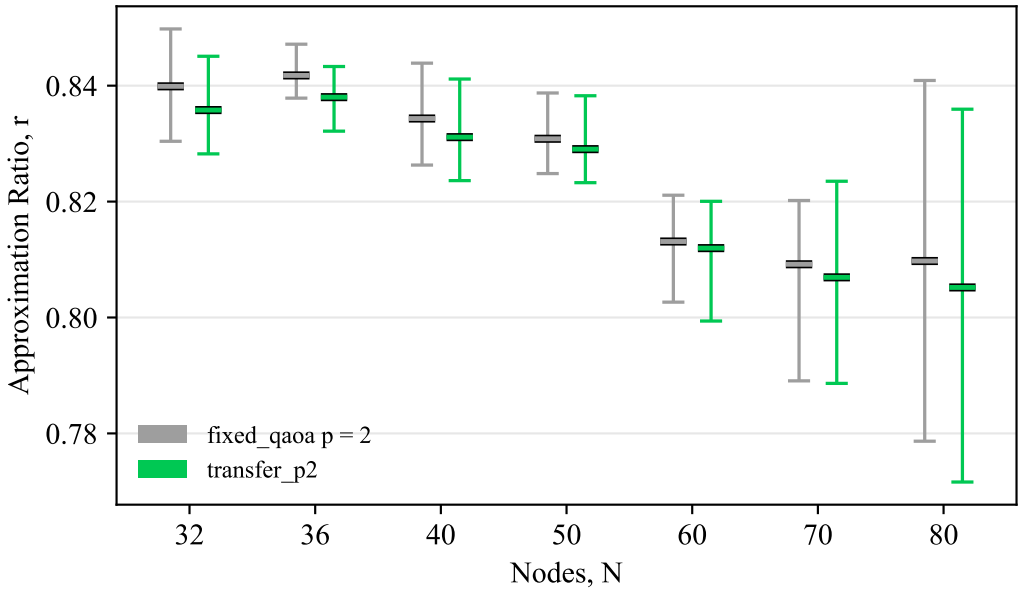

Figure D.1: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N up to $N = 80$ at circuit depth $p = 2$, simulated using Matrix Product States (MPS, bond dimension $\chi = 512$). We compare direct subgraph transfer (`transfer_p2`) against fixed-angle QAOA (`fixed_qaoa`). Solid lines represent mean values across 20 independent random 3-regular graph seeds, and whiskers indicate the Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

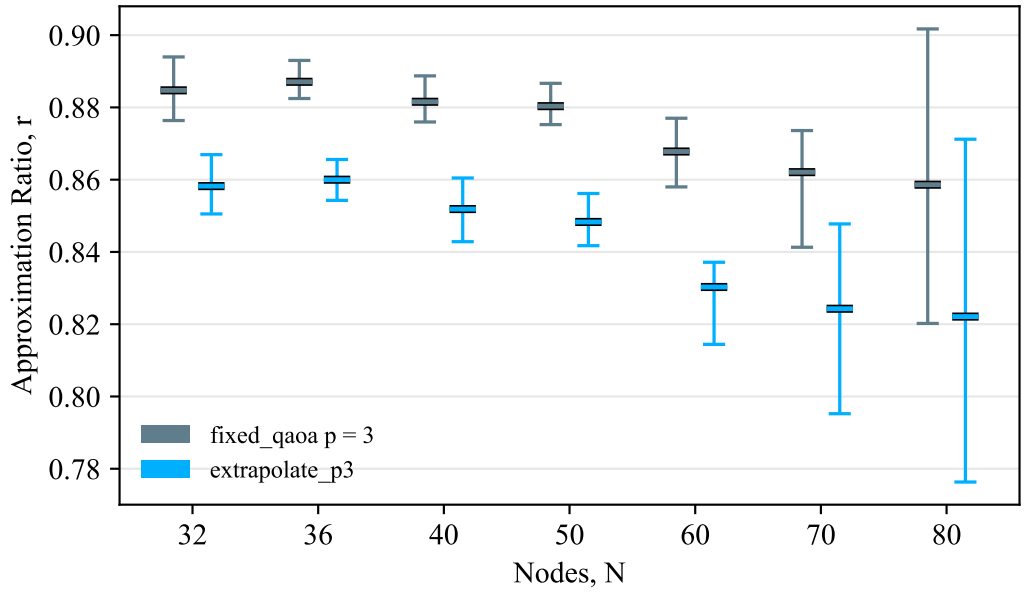

Figure D.2: Approximation ratio $r = \frac{\langle C \rangle}{C^*}$ (Equation 15) as a function of graph size N up to $N = 80$ at circuit depth $p = 3$, simulated using Matrix Product States (MPS, bond dimension $\chi = 512$). We compare linear-ramp extrapolated subgraph transfer (`extrapolate_p3`) against fixed-angle QAOA (`fixed_qaoa`). Solid lines show mean values across 20 independent seeds, and whiskers indicate Q_1 and Q_3 quartiles. Symbols corresponding to the same N are slightly offset horizontally for clarity.

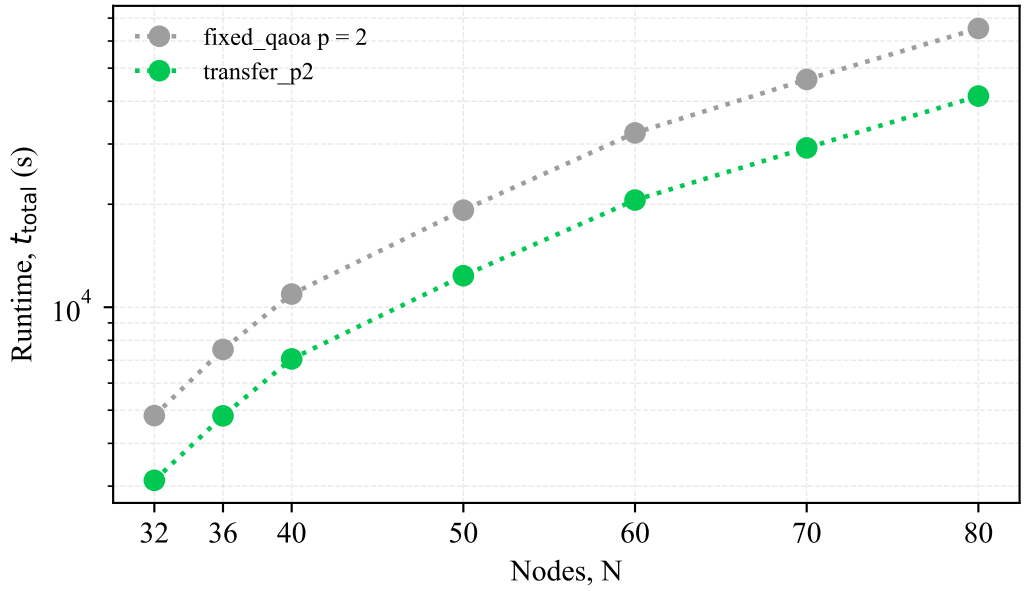

Figure D.3: Total wall-clock runtime (in seconds) as a function of graph size N up to $N = 80$ at circuit depth $p = 2$ for `transfer_p2` and `fixed_qaoa` using the MPS backend ($\chi = 512$). Runtimes are averaged over 20 independent seeds on a single thread.

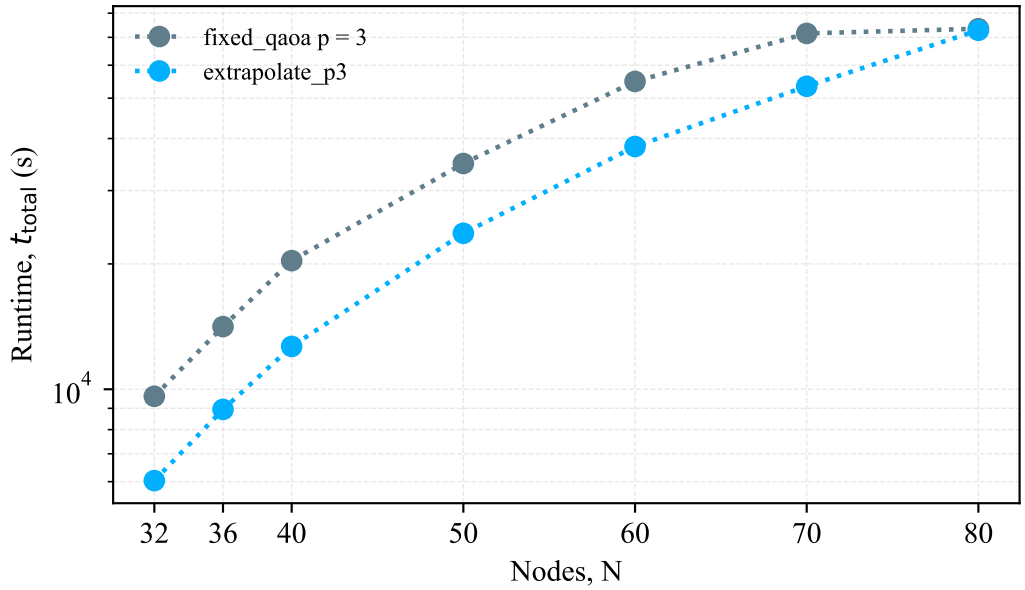

Figure D.4: Total wall-clock runtime (in seconds) as a function of graph size N up to $N = 80$ at circuit depth $p = 3$ for `extrapolate_p3` and `fixed_qaoa` using the MPS backend ($\chi = 512$). Runtimes are averaged over 20 independent seeds on a single thread.

Bibliography

1. A little bit of theory [Internet]. 2014. Available from: **https://acrogenesis.com/or-tools/documentation/user_manual/manual/introduction/theory.html#id24**
2. Bloch Sphere [Internet]. 2012. Available from: **https://commons.wikimedia.org/wiki/File:Bloch_Sphere.svg**
3. Max-cut [Internet]. 2009. Available from: **https://commons.wikimedia.org/wiki/File:Max-cut.svg**
4. The quantum approximate optimization algorithm: optimization problems and implementations. Chalmers Tekniska Hogskola (Sweden); 2023.
5. Towards a Linear-Ramp QAOA Protocol: Evidence of a Scaling Advantage in Solving Some Combinatorial Optimization Problems. npj Quantum Information. 2025 Aug ;11(1):131. doi:**10.1038/s41534-025-01082-1**
6. Graph Theory – Graph Isomorphism. 2026.
7. Drug design on quantum computers. Nature Physics. 2024 ;20(4):549–57.
8. A hybrid quantum computing pipeline for real world drug discovery. Scientific Reports. 2024 ;14(1):16942.
9. The next revolution in computational simulations: Harnessing AI and quantum computing in molecular dynamics. Current Opinion in Structural Biology. 2024 ;89:102919.
10. Quantum mechanics in drug discovery: a comprehensive review of methods, applications, and future directions. International Journal of Molecular Sciences. 2025 ;26(13):6325.
11. Quantum annealing-assisted lattice optimization. npj Computational Materials. 2025 ;11(1):4.
12. Harnessing quantum power: Revolutionizing materials design through advanced quantum computation. Materials Genome Engineering Advances. 2024 ;2(4):e73.
13. Quantum computation of the electronic structure of some prototype solids. Scientific Reports. 2025 ;15(1):44074.
14. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review. 1999 ;41(2):303–32.
15. Quantum machine learning: Classifications, challenges, and solutions. Journal of Industrial Information Integration. 2024 ;42:100736.
16. Classification with quantum machine learning: A survey. arXiv preprint arXiv:200612270. 2020.
17. Quantum computing in artificial intelligence: A review of quantum machine learning algorithms. Path of Science. 2025 ;11(5):7001–9.
18. Quantum computing in logistics and supply chain management an overview. arXiv preprint arXiv:240217520. 2024.

19. Improving the solving of optimization problems: A comprehensive review of quantum approaches. Quantum Reports. 2025 ;7(1):3.

20. Quantum Scheduling Optimization for Airline and Space Missions: A Hybrid Quantum-Classical Approach. Authorea Preprints. 2025.

21. Application of quantum approximate optimization algorithm to job shop scheduling problem. European Journal of Operational Research. 2023 ;310(2):518–28.

22. Ising Formulations of Many NP Problems. Frontiers in Physics. 2014 Feb ;2. doi:**10.3389/fphy.2014.00005**

23. Quantum hardware devices (QHDs): opportunities and challenges. IEEE Access. 2025.

24. Setting the Record Straight on Quantum Computing and RSA Encryption [Internet]. 2024. Available from: **https://www.rsa.com/resources/blog/zero-trust/setting-the-record-straight-on-quantum-computing-and-rsa-encryption/**

25. Variational quantum algorithms. Nature Reviews Physics. 2021 ;3(9):625–44.

26. A Quantum Approximate Optimization Algorithm. arXiv; 2014. doi:**10.48550/arXiv.1411.4028**

27. Barren plateaus in quantum neural network training landscapes. Nature Communications. 2018 Mar ;9(1):4812. doi:**10.1038/s41467-018-07090-4**

28. Multi-Angle Quantum Approximate Optimization Algorithm. Scientific Reports. 2022 Apr ;12(1):6781. doi:**10.1038/s41598-022-10555-8**

29. Performance Analysis of Multi-Angle QAOA for $$p > 1$$. Scientific Reports. 2024 Aug ;14(1):18911. doi:**10.1038/s41598-024-69643-6**

30. Improved Performance of Multi-Angle Quantum Approximate Optimization Algorithm (Ma-QAOA) Compared to QAOA On Simulation and Experimental Hardware Platforms. In: 2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS). 2025. pp. 1130–5. doi:**10.1109/COMSNETS63942.2025.10885567**

31. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM). 1995 ;42(6):1115–45.

32. Training variational quantum algorithms is NP-hard. Physical review letters. 2021 ;127(12):120502.

33. Lecture Notes on Quantum Computing. arXiv; 2025. doi:**10.48550/arXiv.2311.08445**

34. Quantum Computation and Quantum Information. Cambridge University Press; 2000.

35. Quantum Computation [Internet]. 2023. Available from: **http://theory.caltech.edu/~preskill/ph229/**

36. An undulatory theory of the mechanics of atoms and molecules. Physical review. 1926 ;28(6):1049.

37. Symmetry-Informed Transferability of Optimal Parameters in the Quantum Approximate Optimization Algorithm. Physical Review A. 2025 Feb ;111(2):22418. doi:**10.1103/PhysRevA.111.022418**

38. MAXCUT QAOA Performance Guarantees for p >1. Physical Review A. 2021 Apr ;103(4):42612. doi:**10.1103/PhysRevA.103.042612**

39. Analyzing the Quantum Approximate Optimization Algorithm: Ans \"atze, Symmetries, and Lie Algebras. PRX Quantum. 2025 Nov ;6(4):40345. doi:**10.1103/yfwq-yqmk**

40. For fixed control parameters the quantum approximate optimization algorithm's objective function value concentrates for typical instances. arXiv preprint arXiv:181204170. 2018.

41. Transfer Learning of Optimal QAOA Parameters in Combinatorial Optimization. Quantum Information Processing. 2025 May ;24(5):129. doi:**10.1007/s11128-025-04743-4**

42. Augmenting QAOA Ansatz with Multiparameter Problem-Independent Layer. arXiv.

43. Layerwise Retraining and Freezing for Multi-Angle QAOA. Quantum Machine Intelligence. 2026 Jan ;8(1):5. doi:**10.1007/s42484-026-00357-w**

44. Quantum alternating operator ansatz (QAOA) phase diagrams and applications for quantum chemistry. arXiv preprint arXiv:210813056. 2021.

45. Shot noise — Wikipedia, The Free Encyclopedia [Internet]. 2023. Available from: **https://en.wikipedia.org/wiki/Shot_noise**

46. Training the quantum approximate optimization algorithm without access to a quantum processing unit. Quantum Science & Technology. 2020 ;5(3):34008.

47. Graph isomorphism in quasipolynomial time. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing. 2016. pp. 684–97.

48. A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsiya. 1968 ;2(9):12–6.