

Classical Simulation of Quantum Circuits using Tensor Networks: Matrix Product States (MPS) for Quantum Approximate Optimization Algorithms (QAOA)

Annwoy Roy Choudhury

BS-MS Student in Physics (2022-2027)
Indian Institute of Science Education and Research (IISER), Mohali, Punjab, India
Research Internship (May-July 2025) at the
Center for Optical Quantum Technologies, University of Hamburg, Germany

July 30, 2025

Contents

- 1 Introduction & Motivation
- 2 Theoretical Foundations
- 3 Implementation & Methodology
- 4 Problem, Results & Discussion
- 5 Conclusion & Future Outlook
- 6 References

Section 1

Introduction & Motivation

Hopes & Problems

- Quantum computing holds immense promise for solving problems intractable for classical computers.
- Many-body quantum systems are central to advancements in material science, drug discovery, and optimization.
- However, simulating these systems classically faces a fundamental challenge: the **exponential growth** of the Hilbert space.
- This internship explores classical simulation techniques to tackle this challenge, focusing on Quantum Approximate Optimization Algorithms (QAOA).

The Challenge of Quantum Simulation

- A quantum system with N qubits is described by a state vector:

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_N} C_{\sigma_1, \dots, \sigma_N} |\sigma_1, \dots, \sigma_N\rangle$$

- **Exponential Scaling:**
 - Storing full state vector: $O(2^N)$ memory.
 - Manipulating it (e.g., gates): $O(2^N)$ computational cost.
- **Intractability:** Exact simulation impossible for $N > 40 - 50$ qubits on classical computers.
- This limits us and motivates efficient approximate methods.

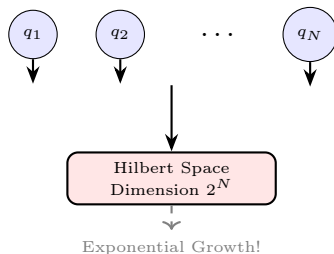
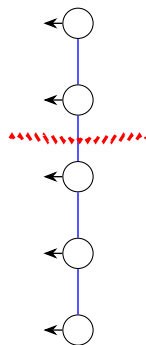


Figure 1: Hilbert space dimension scales exponentially with number of qubits.

Tensor Networks as a Solution

- **Efficient Representation:** Tensor Network (TN) methods provide a powerful framework for efficiently representing and manipulating certain classes of quantum states.
- **Matrix Product States (MPS):** For 1D quantum systems, MPS are particularly effective.
 - They approximate the complex, high-dimensional state vector as a product of smaller tensors (matrices).
 - Their efficiency stems from the "Area Law" for entanglement, common in gapped 1D systems.
- **Bridging the Gap:** MPS enable classical simulation of intractable quantum circuits within reasonable tolerance.



Entanglement scales
with boundary

Figure 2: MPS: entanglement by bond indices.

Internship Objectives

This research internship aimed to gain a comprehensive understanding and practical experience in:

- ➊ **Theoretical Foundations:** Deep dive into Matrix Product States (MPS) and Quantum Approximate Optimization Algorithms (QAOA).
- ➋ **Practical Implementation:** Develop a Python framework for simulating quantum circuits using MPS, including efficient **Gate Application**.
- ➌ **QAOA Simulation:** Apply the developed framework to simulate QAOA circuits for optimization problems (e.g., Max-Cut).
- ➍ **Analysis of Performance:** Study entanglement growth, the impact of compression techniques (SVD, Variational), and fidelity tracking in classical simulations.
- ➎ **Benchmarking:** Explore the boundaries of classical simulability and the trade-offs between accuracy and computational cost for quantum algorithms, effectively the bounds of quantum supremacy.

Section 2

Theoretical Foundations

Defining the Matrix Product State

- Classically simulating quantum systems faces the "Curse of Dimensionality."
- For N qubits, the full state vector lives in a 2^N -dimensional Hilbert space:

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_N=0}^{d-1} C_{\sigma_1, \dots, \sigma_N} |\sigma_1, \dots, \sigma_N\rangle$$

- **Matrix Product States (MPS)** offer an efficient, compressed representation for 1D quantum systems:

$$C_{\sigma_1, \dots, \sigma_N} \approx A_{\alpha_0, \alpha_1}^{\sigma_1} A_{\alpha_1, \alpha_2}^{\sigma_2} \dots A_{\alpha_{N-1}, \alpha_N}^{\sigma_N}$$

- **Physical Indices (σ_i):** Represent the local quantum state (e.g., spin-up/down for a qubit).
 - Dimension: d (e.g., $d = 2$ for qubits).
- **Virtual/Bond Indices (α_i):** Internal indices that are contracted (summed over).
 - Carry entanglement information between parts of the system.
 - Dimension: χ (or D), known as the **bond dimension**.

Diagrammatic Representation: A Chain of Tensors

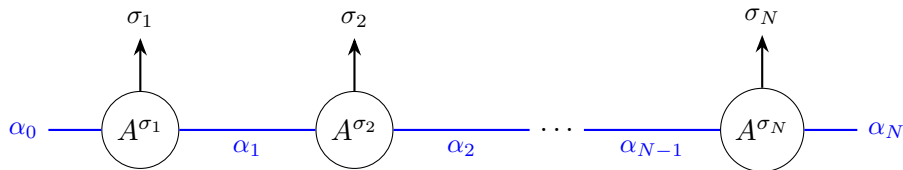


Figure 3: Diagram of a Matrix Product State (MPS). Each circle is a tensor A^{σ_i} with a physical index (σ_i) and virtual/bond indices (α_{i-1}, α_i) connecting it to neighbors.

- Circles represent individual tensors (generalized matrices), one per site (qubit).
- Black arrows (σ_i) are **physical indices**: specify the local quantum state.
- Blue lines (α_i) are **virtual/bond indices**: contracted between tensors, they encode quantum correlations and entanglement.
- The **bond dimension** (χ or D) is the dimension of these virtual links, controlling the MPS's expressive power.

Deconstructing Complex Tensors

- Tensor decomposition is the process of breaking down a large, complex tensor into a product of simpler, structured tensors.
- This is analogous to matrix factorization (e.g., $M = LU$ decomposition).
- It's **fundamental** to creating, manipulating, and compressing MPS.

Singular Value Decomposition (SVD): The Cornerstone

- SVD is arguably the most important decomposition in tensor network analysis, often called "rank-revealing."
- For any matrix M , SVD provides a factorization:

$$M = USV^\dagger$$

- U : Isometric matrix (columns are orthonormal).
- S : Diagonal matrix of non-negative real numbers, the **singular values** (s_α).
- V^\dagger : Conjugate transpose of an isometric matrix.

Schmidt Decomposition: Bipartite Entanglement and Singular Values

- For any pure quantum state $|\Psi\rangle$ of a bipartite system $(A + B)$, it can be uniquely written as:

$$|\Psi\rangle = \sum_{\alpha=1}^r s_{\alpha} |\phi_{\alpha}\rangle_A |\chi_{\alpha}\rangle_B$$

- s_{α} : Non-negative real numbers, the **Schmidt coefficients** (singular values).
 - $|\phi_{\alpha}\rangle_A, |\chi_{\alpha}\rangle_B$: Orthonormal basis states for A and B.
 - r : **Schmidt rank** (number of non-zero s_{α}).
- Meaning:**
 - If $r = 1$, it's a **product state** (no entanglement).
 - If $r > 1$, it's **entangled**. Higher r = more entanglement.



Figure 4: Bipartition of a quantum system into A and B.

Quantifying Entanglement & Optimal Compression

- **Entanglement Diagnosis:**

- Singular values (s_α) = Schmidt coefficients.
- Their decay pattern reveals entanglement.

- **Optimal Compression:**

- Keep k largest s_α for best approximation (Eckart-Young Theorem).
- Foundation of MPS compression.

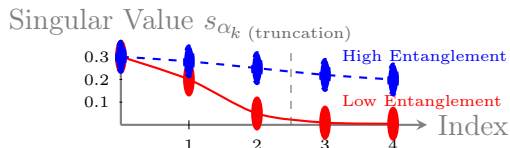


Figure 5: Singular value decay for different entanglement levels, illustrating truncation.

QR and RQ Decompositions

QR Decomposition ($M = QR$):

- Q : Isometric. R : Upper-triangular.
- **Use Case:** Left-to-right canonicalization.
- Faster than SVD.

RQ Decomposition ($M = RQ$):

- R : Lower-triangular. Q : Isometric.
- **Use Case:** Right-to-left canonicalization.
- Faster than SVD.

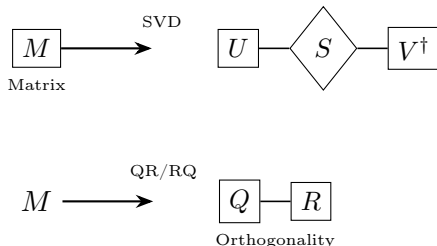


Figure 6: Comparison of SVD vs. QR/RQ.

Key Trade-off:

- **SVD is "Smart":** Provides entanglement info.
- **QR/RQ are "Handy":** Faster, no entanglement info.

Generalized Decompositions by Mode

- Tensor decompositions (SVD, QR) are applied to higher-rank tensors by selecting a specific index, called the "**mode**".
- The tensor is then **reshaped** into an effective 2D matrix, allowing standard matrix factorizations.
- This process is fundamental for breaking down complex MPS tensors into simpler components.

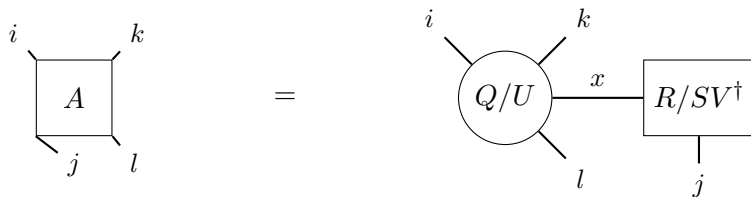


Figure 7: An arbitrary tensor A decomposed by isolating mode j into an isometry (Q/U) and a remainder (R/SV^\dagger), connected by a new bond index x .

Gauge Freedom: The Ambiguity

- A "random" MPS representation is **not unique**.
- We can insert an invertible matrix X and its inverse X^{-1} on any bond:

$$\dots M_{\alpha_{i-1}, \alpha_i}^{\sigma_i} M_{\alpha_i, \alpha_{i+1}}^{\sigma_{i+1}} \dots = \dots (M^{\sigma_i} X)_{\alpha_{i-1}, \alpha'_i} (X^{-1} M^{\sigma_{i+1}})_{\alpha'_i, \alpha_{i+1}} \dots$$

- This transformation doesn't change the physical quantum state, but it alters the individual tensors ($M^{\sigma_i} \rightarrow M^{\sigma_i} X$, and $M^{\sigma_{i+1}} \rightarrow X^{-1} M^{\sigma_{i+1}}$).
- This "gauge freedom" makes it challenging to work with MPS directly or uniquely define properties.

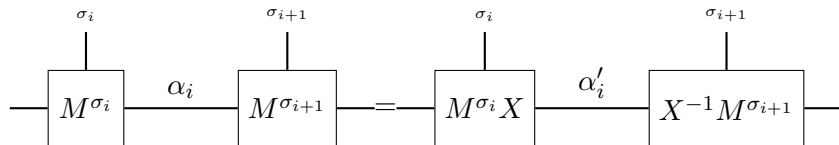


Figure 8: Inserting XX^{-1} on a bond changes individual tensors but not the total state.

Canonical Forms: Imposing Orthogonality

- To simplify calculations and reveal entanglement, MPS tensors are transformed into **canonical forms**.
- These forms impose specific orthogonality properties on the tensors.

Left-Canonical Form (A-tensors):

- All tensors to the left are "left-normalized."
- Property: $\sum_{\sigma, \alpha'} (A_{\alpha', \alpha}^{\sigma})^* A_{\alpha', \alpha}^{\sigma} = \delta_{\alpha, \alpha'}$

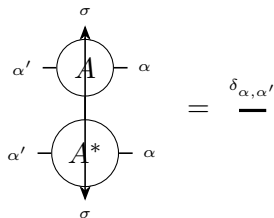


Figure 9: Left-canonical ($A^\dagger A = I$).

Right-Canonical Form (B-tensors):

- All tensors to the right are "right-normalized."
- Property: $\sum_{\sigma, \alpha} B_{\alpha', \alpha}^{\sigma} (B_{\alpha', \alpha}^{\sigma})^* = \delta_{\alpha', \alpha'}$

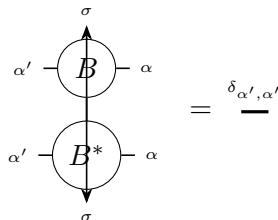


Figure 10: Right-canonical ($BB^\dagger = I$).

Mixed-Canonical Form

- An MPS is in **Mixed-Canonical Form** when tensors to the left are left-canonical (A's) and tensors to the right are right-canonical (B's).
- The "unnormalized" tensor in the middle is the **Orthogonality Center**.
- It holds all the norm and entanglement information for that specific "cut" in the chain.
- **Utility:** Essential for efficient calculations (e.g., expectation values) and for algorithms that sweep through the MPS (like DMRG).

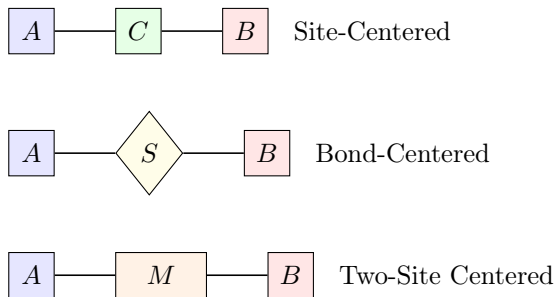


Figure 11: Types of orthogonality centers in an MPS chain (A = Left-canonical, B = Right-canonical).

The Vidal Gauge

- The **Vidal Gauge** (also known as Γ -form) is a specialized canonical form of MPS.
- It explicitly separates the isometric tensors (Γ) from the singular values (Λ).
- The singular values (Λ_i) live directly on the virtual bonds between the Gamma tensors.
- **Utility:** This form is very convenient for:
 - Directly reading off Schmidt coefficients for any bond.
 - Efficiently applying diagonal operators to bonds.
 - Many advanced MPS algorithms (e.g., TEBD, DMRG).

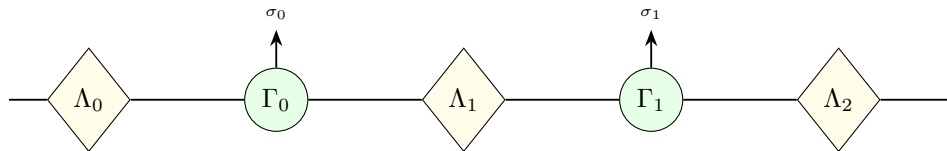


Figure 12: A Matrix Product State in Vidal Gauge

Tensor Contraction

- **Tensor Contraction:** The fundamental operation for combining multiple tensors.
- It involves summing over common (contracted) indices shared between tensors.
- This process reduces the number of indices and combines information, resulting in a new tensor.
- It's the core primitive for all MPS manipulations and calculations.

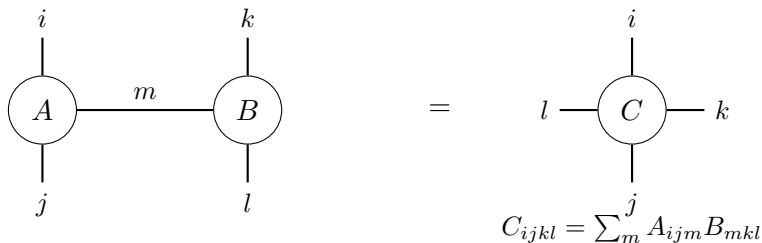


Figure 13: Tensor contraction: Summing over common indices to form a new tensor.

Tensor Contraction Cost

Contraction Order Matters!

Naive MPS Contraction (Non-optimal Order):

- Summing all internal indices simultaneously, or choosing an inefficient order.
- Can lead to very large intermediate tensors ($O(\chi^L)$).
- **Example:** Computing $\langle T, W \rangle$ by forming full T and W tensors first:

$$O(d^N)$$

(where d^N is the size of the full tensor)

- **Key Takeaway:** Optimal contraction order transforms an exponentially intractable problem into a polynomially solvable one for 1D systems, enabling simulation of much larger systems.

Zipper Algorithm (Optimal Order):

- An optimized iterative method for contracting entire MPS chains (e.g., inner products).
- Sequentially contracts local tensors and builds environment tensors (e.g., $E_{\beta_i \alpha_i}$).
- **Cost (per site):** $O(\chi^3 \cdot d)$
- **Total Cost (for chain length L):** Scales **polynomially** with L .

$$O(L \cdot \chi^3 \cdot d)$$

Visualizing Contraction Order

Naive Full Tensor Contraction

$$\langle T, W \rangle = \sum_{\{\mathbf{s}\}} T^{s_1 s_2 s_3 s_4 s_5 s_6} W^{s_1 s_2 s_3 s_4 s_5 s_6}$$



Figure 14: Full tensors are constructed and then contracted.

Zipper Algorithm (Optimal)

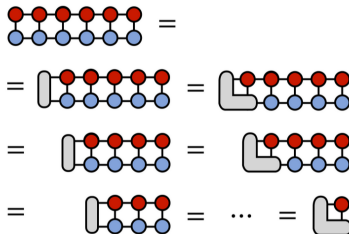


Figure 15: Iterative contraction building environment tensors.

(Diagrams taken from [tensornetwork.org/mps/innerprod](https://tenso.network.org/mps/innerprod))

Constructing an MPS from a Quantum State

- To start many MPS-based simulations, an initial quantum state is represented as an MPS.
- This is achieved via an **iterative Singular Value Decomposition (SVD) procedure**.
- The process "peels off" one site's tensor at a time from the full coefficient tensor.

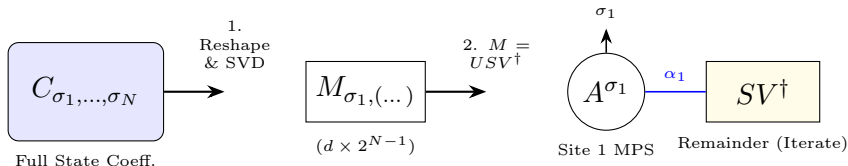


Figure 16: First step of iterative SVD for MPS construction: peeling off Site 1.

Gate Application: Applying Single-Qubit Gates

- Applying a single-qubit gate (U) to a site i in an MPS is computationally simple.
- It does **not change the entanglement structure** of the state, nor the bond dimensions.
- The gate is applied by contracting its matrix elements with the physical index of the MPS tensor at site i .

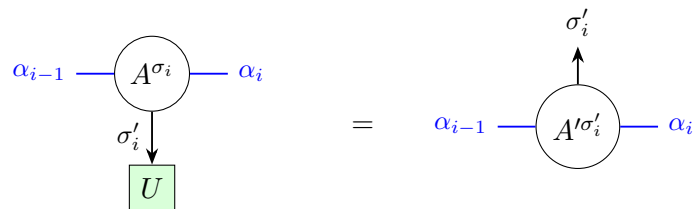

$$(A')_{\alpha_{i-1}, \alpha_i}^{\sigma'_i} = \sum_{\sigma_i} U_{\sigma'_i, \sigma_i} A_{\alpha_{i-1}, \alpha_i}^{\sigma_i}$$

Figure 17: Applying a single-qubit gate U to an MPS tensor A .

Gate Application: Applying Two-Qubit Gates

- Applying a two-qubit gate (U) to neighboring sites (i and $i + 1$) is the **essential step** for generating entanglement in an MPS.
- Unlike single-qubit gates, two-qubit gates typically **increase the entanglement** between the involved sites.
- This demands an increased bond dimension (χ) to accurately represent the new state.

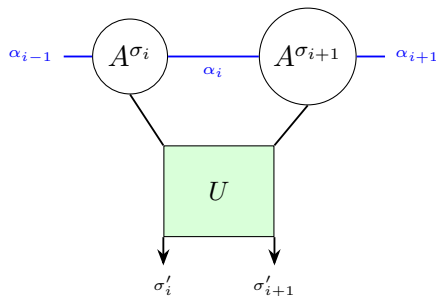


Figure 18: Applying a two-qubit gate U to two adjacent MPS tensors.

Two-Qubit Gate Application Workflow

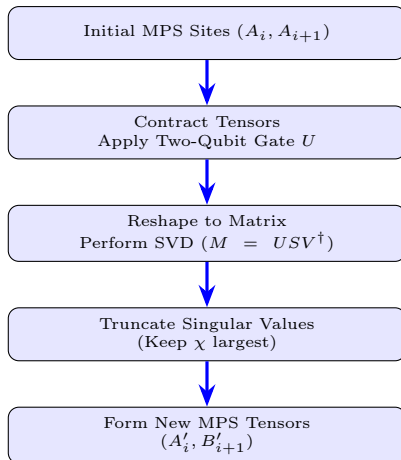


Figure 19: Workflow for applying a two-qubit gate with SVD truncation.

Necessity of MPS Compression

- **Entanglement Growth:** Applying two-qubit gates (and time evolution) continuously increases the entanglement in the MPS.
 - This causes the required bond dimension (χ) to grow.
 - For exact simulations, χ can grow **exponentially** with circuit depth p (e.g., $\chi \sim 2^p$).
- **Computational Bottleneck:** An exponentially growing χ makes the simulation intractable, even with the efficient $O(L\chi^3d)$ Zipper Algorithm.
- **The Solution: Compression!**
 - We "compress" the MPS back to a fixed, manageable maximum bond dimension (χ_{\max}) after each step or batch of gates.
 - This trades **perfect accuracy** for **computational tractability**.
- Two main strategies: **SVD Compression** (greedy) and **Variational Compression** (optimal).

SVD Compression

- **Core Idea:** Direct, non-iterative method to reduce bond dimension (χ).
- Leverages SVD to identify and keep only the most important components.

How it Works:

- ➊ After gate application, form uncompressed local tensor/matrix M .
- ➋ Perform SVD: $M = USV^\dagger$.
- ➌ **Truncate:** Keep only χ_{\max} largest singular values from S .
- ➍ Reconstruct new, compressed MPS tensors from U and truncated $S_{\chi_{\max}} V^\dagger$.

Characteristics:

- **Fast & Direct:** Non-iterative.
- **Locally Optimal:** Provides the best possible approximation at each individual bond (Eckart-Young Theorem).
- **Globally Suboptimal (Greedy):** Local truncation errors can accumulate across the entire MPS chain.

Variational Compression

- **Core Idea:** Directly optimizes global fidelity between target and compressed MPS.
- Equivalent to minimizing global error $|||\Psi\rangle - |\Phi\rangle||^2$.
- Inspired by Density Matrix Renormalization Group (DMRG) algorithm.

How it Works:

- 1 **Initialization:** Target MPS and initial guess MPS (e.g., from SVD comp.).
- 2 **Sweeping:** Iteratively optimize one tensor, building "environment tensor" from rest of network. Optimal update is proportional to environment.
- 3 **Convergence:** Repeat sweeps (left-to-right, then right-to-left) until global fidelity converges.

Characteristics:

- **Globally Optimal:** Finds local minimum, typically the true optimum.
- **Highly Accurate:** State-of-the-art for precision at fixed bond dimension.
- **Slower & Iterative:** More computationally demanding than SVD compression.

Visualizing Compression Algorithms

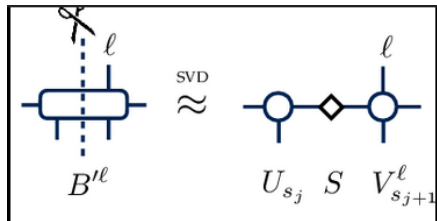


Figure 20: SVD Compression (From "Supervised Learning with Quantum-Inspired Tensor Networks" by E. Stoudenmire and D. J. Schwab)

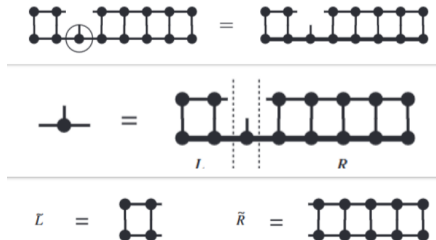


Figure 21: Variational Compression (From "The density-matrix renormalization group in the age of matrix product states" by U. Schollwöck)

QAOA: From Adiabatic Quantum Computation (AQC)

- QAOA is inspired by **Adiabatic Quantum Computation (AQC)**, a beautiful but often impractical method.
- **AQC Dream:** To solve optimization problems by finding the ground state of a problem Hamiltonian (H_C).
- **AQC Process:**
 - 1 Start in ground state of simple initial Hamiltonian (H_B).
 - 2 Slowly transform $H_B \rightarrow H_C$ over a long time T .
 - 3 Adiabatic Theorem guarantees system stays in ground state.

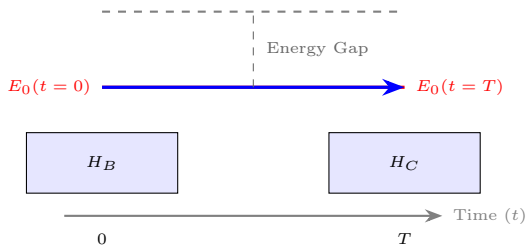


Figure 22: Adiabatic Evolution: continuous transformation from H_B to H_C .

Quantum Approximate Optimization Algorithm (QAOA)

1. Cost Hamiltonian (H_C):

- Encodes optimization problem in energy spectrum.
- Eigenvalues = cost of basis states.
- **Role:** Applies phase \propto cost (Phase Separation).
- Diagonal in computational basis (e.g., $Z_i Z_j$).

Example (Max-Cut on 2 nodes):

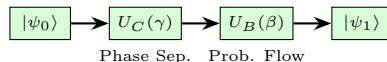
$$H_C = Z_0 Z_1$$

- $|01\rangle, |10\rangle$: Energy -1 (Good)
- $|00\rangle, |11\rangle$: Energy +1 (Bad)

2. Mixer Hamiltonian (H_B):

- Drives exploration of solution space.
- Facilitates "probability flow" & interference.
- **Role:** Mixes amplitudes based on H_C phases (Interference).
- Typically sum of Pauli-X ops (e.g., $X_0 + X_1$).
- Non-diagonal in computational basis.

Figure 23: A single QAOA layer: H_C phases, H_B mixes amplitudes.



QAOA Circuit Overview

- The full QAOA algorithm prepares a state by repeating the two-step (Cost-Mixer) process for p "layers."
- The integer p is the **depth** of the algorithm.
- The final state is obtained by applying the layered operator to an initial uniform superposition state $|+\rangle^{\otimes N}$.

The QAOA Ansatz:

$$|\psi(\vec{\gamma}, \vec{\beta})\rangle = \prod_{k=1}^p e^{-i\beta_k H_B} e^{-i\gamma_k H_C} |+\rangle^{\otimes N}$$

- $\vec{\gamma} = (\gamma_1, \dots, \gamma_p)$ and $\vec{\beta} = (\beta_1, \dots, \beta_p)$ are the $2p$ variational parameters.

Hamiltonian Operators:

- **Cost Hamiltonian** (H_C): Diagonal, encodes problem.

$$H_C = \sum_{\langle i,j \rangle} J_{ij} Z_i Z_j + \sum_i h_i Z_i$$

- **Mixer Hamiltonian** (H_B): Non-diagonal, creates superpositions.

$$H_B = \sum_i X_i$$

QAOA Circuit Visualization

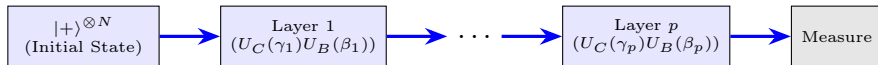


Figure 24: General structure of a QAOA circuit with p layers.

QAOA Problem Mapping to Quantum Systems: The Max-Cut Problem

- Many classical combinatorial optimization problems can be mapped to a form suitable for quantum algorithms like QAOA.
- **Max-Cut Problem:** Partition the vertices of a graph $G = (V, E)$ into two sets (0 and 1) to maximize the number of edges connecting vertices in different sets.

Variables:

- Assign binary $x_i \in \{0, 1\}$ to each vertex $i \in V$.
- $x_i = 0$ (Set 0), $x_i = 1$ (Set 1).

Cost Function Logic:

- An edge $(i, j) \in E$ is "cut" if $x_i \neq x_j$.
- Simple expression for cut edge:
 $(x_i - x_j)^2$:
 - If $x_i = x_j$ (not cut): $(x_i - x_j)^2 = 0$.
 - If $x_i \neq x_j$ (cut): $(x_i - x_j)^2 = 1$.

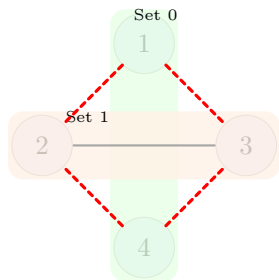


Figure 25: Max-Cut example with partition.

Quadratic Unconstrained Binary Optimization (QUBO)

- Standardized format for translating classical optimization problems to quantum systems.
- Aims to minimize a quadratic function of binary variables:

$$\min_{\mathbf{x} \in \{0,1\}^N} C(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$$

- $\mathbf{x} \in \{0,1\}^N$: N binary variables.
- Q : Matrix of coefficients (Q_{ii} for linear costs, Q_{ij} for quadratic interactions).

Key Properties of QUBO:

- **Binary:** Variables are 0 or 1.
- **Unconstrained:** All logic/penalties incorporated into $C(\mathbf{x})$.
- **Quadratic:** Cost function is degree two.

Universality: Many NP-hard problems can be formulated as QUBO.

From QUBO to the Ising Hamiltonian

- The bridge to quantum algorithms: QUBO problems are equivalent to finding the ground state of an **Ising Model**.
- A Cost Hamiltonian (H_C) is constructed whose energy spectrum directly reflects classical QUBO costs.

The Mathematical Mapping: Binary to Pauli-Z

- Map classical binary variables ($x_i \in \{0, 1\}$) to quantum Pauli-Z operators ($Z_i \in \{-1, +1\}$).
- **Transformation Equation:**

$$x_i = \frac{I - Z_i}{2}$$

- If qubit i is in state $|0\rangle \implies Z_i = +1 \implies x_i = 0$.
- If qubit i is in state $|1\rangle \implies Z_i = -1 \implies x_i = 1$.

This transformation ensures the quantum Hamiltonian's eigenvalues align with classical costs.

Resulting Ising Hamiltonian (H_C)

- By substituting $x_i = \frac{1-Z_i}{2}$ into the general QUBO form ($C(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$), and re-arranging terms, we obtain the Cost Hamiltonian H_C .
- This H_C is always **diagonal in the computational basis**.

General Form of the Ising Hamiltonian (H_C):

$$H_C = \sum_i h_i Z_i + \sum_{i < j} J_{ij} Z_i Z_j$$

- $\sum_i h_i Z_i$ (**1-local terms** / "local fields"):
 - Result from original linear terms (x_i) and diagonal terms (x_i^2) in QUBO.
 - Represent a local bias on each qubit.
- $\sum_{i < j} J_{ij} Z_i Z_j$ (**2-local terms** / "couplings"):
 - Result from original quadratic interaction terms ($x_i x_j$) in QUBO.
 - Represent interaction strengths between pairs of qubits.

Higher-order terms ($Z_i Z_j Z_k$) can also arise from more complex mappings (e.g., PUBO problems).

From QAOA Problem to Quantum Circuit on MPS

- Classical optimization problems are mapped to QAOA Hamiltonians (H_C, H_B).
- Quantum states are represented efficiently using Matrix Product States (MPS).
- The connection: how QAOA's Hamiltonian evolutions act on an MPS.

The Full Pipeline (Conceptually):

- **Classical Problem** $\xrightarrow{\text{Map}}$ **QAOA Hams.** (H_C, H_B)
- **QAOA Hams.** $\xrightarrow{\text{Decompose}}$ **QAOA Circuit Ops.** (U_C, U_B)
- **QAOA Circuit Ops.** $\xrightarrow{\text{Simulate Via}}$ **Matrix Product State (MPS)**

Key Connection:

- Hamiltonian evolution operators (e.g., U_C) are implemented as sequences of **local quantum gates**.
- These local gates act directly on the MPS tensors.
- This enables efficient QAOA simulation by leveraging MPS's compressed structure.

Section 3

Implementation & Methodology

Overview of the Python Framework

- Developed a modular Python framework for classical quantum circuit simulation using MPS.
- Relies heavily on NumPy and SciPy for efficient tensor operations.

Framework Modules:

- **General Tensor Functions:** SVD, QR, RQ decompositions.
- **Basic MPS Functions:** Canonicalization, norm/amplitude, inner products (Zipper Algo.), svd compression, variational compression.
- **MPS Gate Applications:** Single-qubit and two-qubit gate workflows on MPS.
- **QAOA Simulation & Analysis:** Orchestrates full QAOA (exact/inexact methods).

Workflow:

- **Input:** Quantum state or optimization problem.
- **Process:** Data flows from basic tensor ops to complex QAOA simulations.
- **Output:** Optimization results or detailed simulation analysis.

From Math to Code

- Implemented theoretical concepts as Python functions using NumPy.
- Direct translation from math to computational routines.

Key Function Implementations:

● Tensor Decompositions:

- `svd_tensor_mode()`: SVD by isolating a mode.
- `qr_tensor_mode()`: QR decomposition by mode.

● MPS Canonicalization:

- `make_left_canon()`: Left-canonical form.
- `make_mixed_canon()`: Shift orthogonality center.

● MPS Utilities:

- `inner_product()`: Computes inner product via **Zipper Algorithm**.
- `calculate_expectation_value()`: Expectation values.

● MPS Compression:

- `svd_compression_fixed_bond()`: Greedy SVD compression.
- `variational_compression()`: Optimal iterative compression.

Python's Role:

- NumPy provides optimized array manipulation for tensor operations.
- Enables flexible and efficient implementation of TN algorithms.

Simulating Quantum Circuits with MPS

- Python framework integrates MPS functions for quantum circuit simulation (QAOA).
- Key steps: **Gate Application** and the **Hybrid Optimization Loop**.

1. MPS Gate Application:

- Quantum gates apply directly to the MPS state.
- `apply_single_qubit_gate()`: Simple local tensor operation.
- `apply_two_qubit_gate()`: Involves contraction, SVD, and truncation.

2. Hybrid QAOA Optimization Loop:

- **Classical Optimizer:** Proposes parameters $(\vec{\gamma}, \vec{\beta})$.
- **Quantum Circuit (MPS Simulator):** Executes layered QAOA circuit with parameters.
- **Score Calculation:** Computes expectation value of H_C .
- **Feedback:** Optimizer uses score to refine parameters for next iteration.

Exact vs. Inexact Simulation

- **Problem:** Exact classical simulation (no compression) is limited by **exponential bond dimension growth** ($\chi \sim 2^p$).
- This makes it quickly **intractable** for large systems (N) or deep circuits (p).

1. Exact State (Reference for Small Systems):

- For **small** N or p , the exact, unapproximated state (Ψ_{exact}) is computed.
- Used as a **benchmark** to quantify infidelity of compressed simulations:

$$\text{Infidelity} = 1 - |\langle \Psi_{\text{exact}} | \Psi_{\text{inexact}} \rangle|^2$$

- **Limitation:** Ψ_{exact} becomes computationally **unreachable** for large N or p .

2. Inexact Simulation (Practical Approach):

- **Goal:** Achieve polynomial scaling by accepting a controlled error.
- **Method:** Enforce a **fixed max bond dimension** (χ_{max}).
- MPS is **compressed** back to χ_{max} after each operation (SVD or Variational methods).
- **Pros:** Polynomial scaling ($O(L\chi_{\text{max}}^3 d)$), tractable for large systems.
- **Challenge:** How to estimate fidelity when Ψ_{exact} is unknown?

Fidelity Estimation: Quantifying Inexactness

- For large inexact simulations, the true exact state (Ψ_{exact}) is computationally **unreachable**.
- Direct infidelity calculation ($1 - |\langle \Psi_{\text{exact}} | \Psi_{\text{inexact}} \rangle|^2$) is impossible.
- Need a practical method to **estimate accuracy** without knowing Ψ_{exact} .

The Multiplicative Fidelity Law:

- **Core Principle:** Total fidelity (F_{total}) is approximated by the product of "partial fidelities" (f_{δ}) from each compression step.

$$F_{\text{total}} \approx \prod_{\delta=1}^m f_{\delta}$$

- Each $f_{\delta} = |\langle \Psi_{\text{compressed}(\delta)} | \Psi_{\text{target}(\delta)} \rangle|^2$ (fidelity with uncompressed state at that step).
- f_{δ} is calculated **during simulation**.

Significance:

- Provides **accurate, internally consistent estimate** of final state's fidelity.
- Enables tracking of approximation quality when exact answer is unknown.
- Justified by chaotic error evolution (errors become nearly independent).

Section 4

Problem, Results & Discussion

Solving Linear Max-Cut with MPS: Initial Setup

- **QAOA Problem:** Linear Max-Cut.
 - Graph: Nodes in a line, edges connect nearest neighbors.
 - Goal: Partition nodes into two sets to maximize cut edges.
- **Method:** Classical simulation using Matrix Product States (MPS).
 - Implemented with exact gate tensors for initial tests.

Problem Mapping (High-Level):

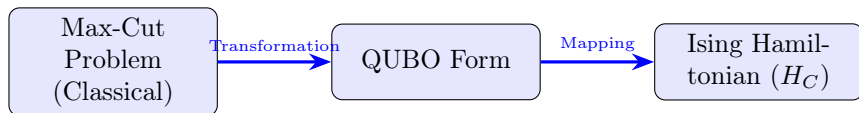


Figure 26: High-level problem mapping for QAOA.

Initial Simulation Purpose:

- Confirmed MPS can accurately solve this problem.
- **Main Focus (next slides):** Investigating **bond dimension growth** and **error accumulation** for different simulation approaches.

Problem Mapping: Mathematical Details

- Max-Cut (maximizing cut edges) is converted to a minimization problem.
- This converts to **Quadratic Unconstrained Binary Optimization (QUBO)** form.

From Max-Cut to QUBO ($C(\mathbf{x})$):

- Cuts for edge (i, j) : $(x_i - x_j)^2 = x_i + x_j - 2x_i x_j$.
- Max-Cut is $\sum_{(i,j) \in E} (x_i + x_j - 2x_i x_j)$.
- For minimization, QAOA Cost Function $C(\mathbf{x})$ is its negative:

$$C(\mathbf{x}) = \sum_{(i,j) \in E} (2x_i x_j - x_i - x_j)$$

From QUBO to Ising Hamiltonian (H_C):

- Map binary $x_i \in \{0, 1\}$ to Pauli-Z $Z_i \in \{-1, +1\}$ via:

$$x_i = \frac{I - Z_i}{2}$$

- Substituting into $C(\mathbf{x})$ yields the Ising Hamiltonian:

$$H_C = \sum_i h_i Z_i + \sum_{i < j} J_{ij} Z_i Z_j$$

Entanglement Growth: Exact Simulation (vs. N)

- Exact simulation: χ grows freely for perfect accuracy.
- Implication: Significant classical simulation bottleneck for increasing system sizes.

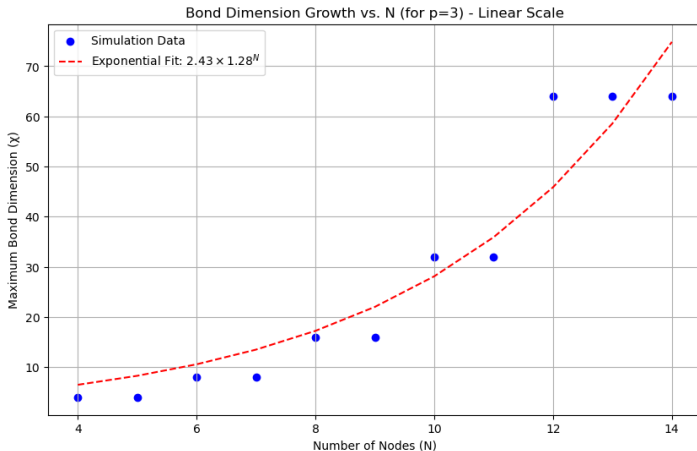


Figure 27: Max Bond Dimension vs. N (for $p=3$).

Entanglement Growth: Exact Simulation (vs. p)

- Exact simulation: χ grows freely as circuit depth (p) increases.
- Implication: Intractability of exact classical simulation for deep quantum circuits, necessitating compression.

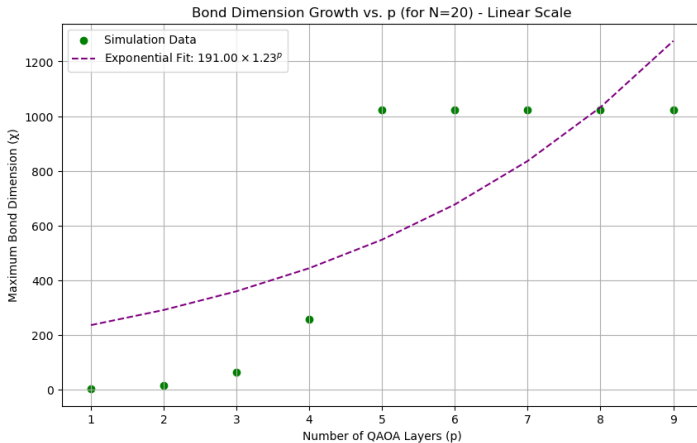


Figure 28: Max Bond Dimension vs. p (for $N=15$).

Entanglement Growth: Exact vs. Compressed (vs. N)

- Compression aims to cap exponential bond dimension growth (χ).
- Comparison: Exact simulation vs. fixed $\chi_{\max} = 4$ compression strategies.

Bond Dimension Growth vs. N (for p=5)

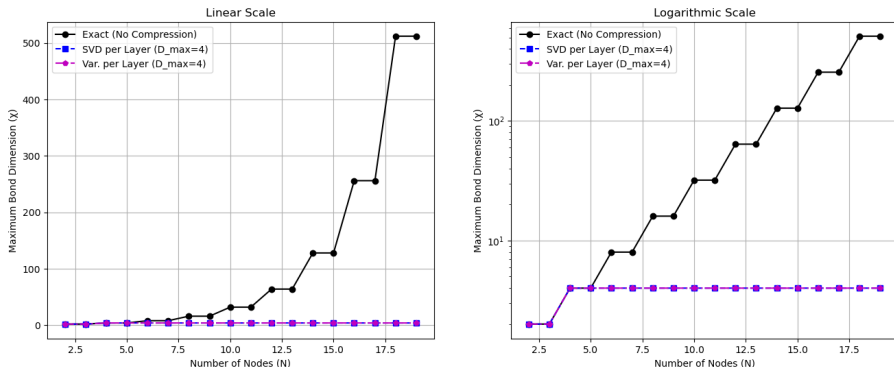


Figure 29: Max Bond Dimension vs. Nodes (N) for p=5.

Entanglement Growth: Exact vs. Compressed (vs. p)

Bond Dimension Growth vs. p (for $N=15$)

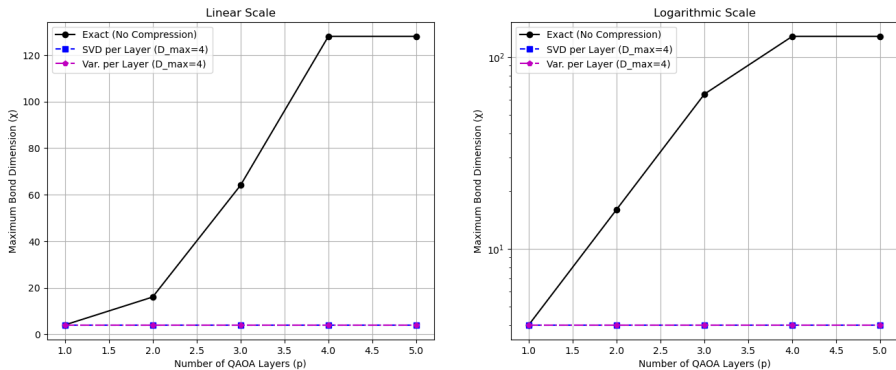


Figure 30: Max Bond Dimension vs. Layers (p) for $N=15$.

Infidelity: Exact Reference – Comparison (vs. N)

- Using Ψ_{exact} reference for infidelity (small N/p).

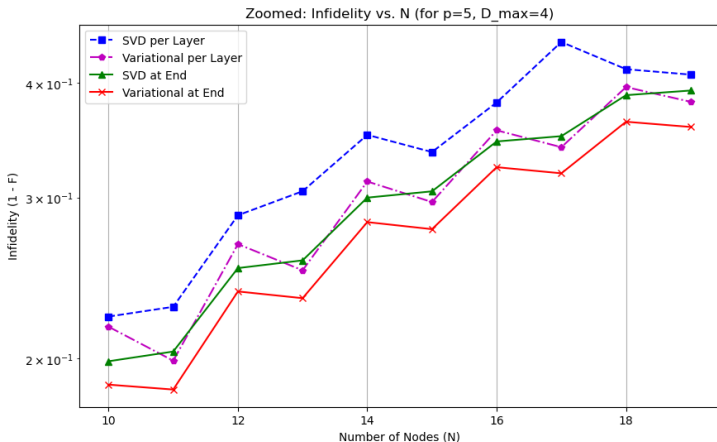


Figure 31: Infidelity vs. N ($p=5$).

- Obs:** Infidelity $\uparrow N$. "Per Layer" \geq "at End". Variational $>$ SVD.

Infidelity: Exact Reference – Comparison (vs. p)

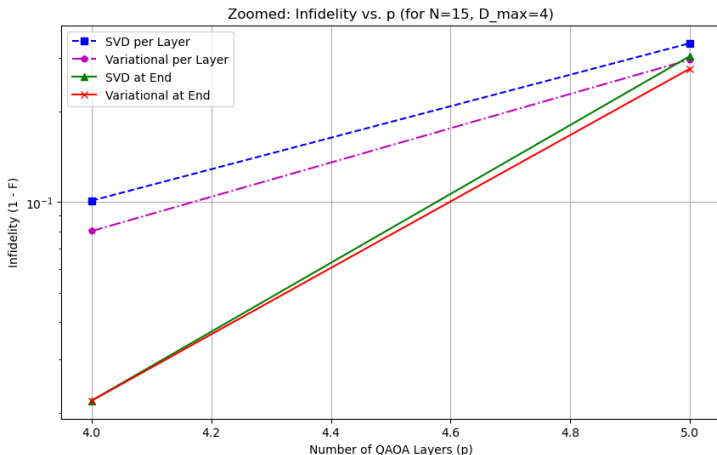


Figure 32: Infidelity vs. p ($N=15$).

● **Obs:** Infidelity $\uparrow p$. "Per Layer" \geq "at End". Variational $>$ SVD.

Validating Inexact Fidelity Estimation

- For large systems, direct infidelity calculation against exact state is impossible.
- We use the **Multiplicative Fidelity Law** to estimate infidelity from inexact simulation.
- This plot validates estimated infidelity vs. true infidelity (for small systems).

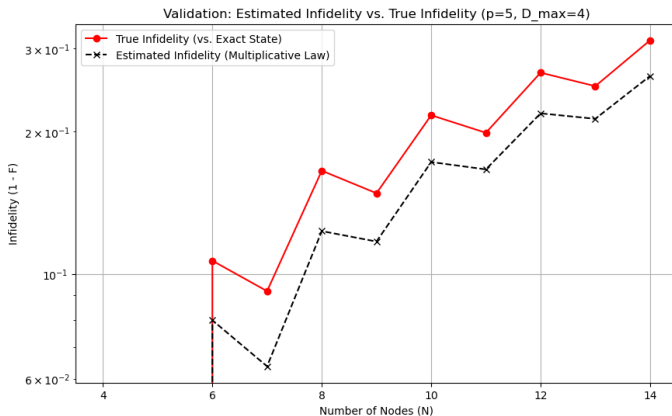


Figure 33: Validation: Estimated Infidelity vs. True Infidelity ($p=5$, $D_{\max}=4$).

Performance of Inexact Simulations (vs. N)

- Evaluate inexact simulation performance using **estimated infidelity** (Multiplicative Fidelity Law).
- Max bond dimension fixed at $\chi_{\max} = 4$.

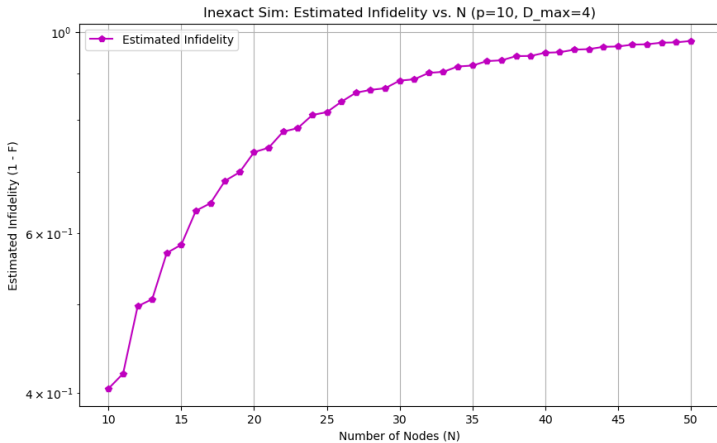


Figure 34: Estimated Infidelity vs. N ($p=10$).

Performance of Inexact Simulations (vs. p)

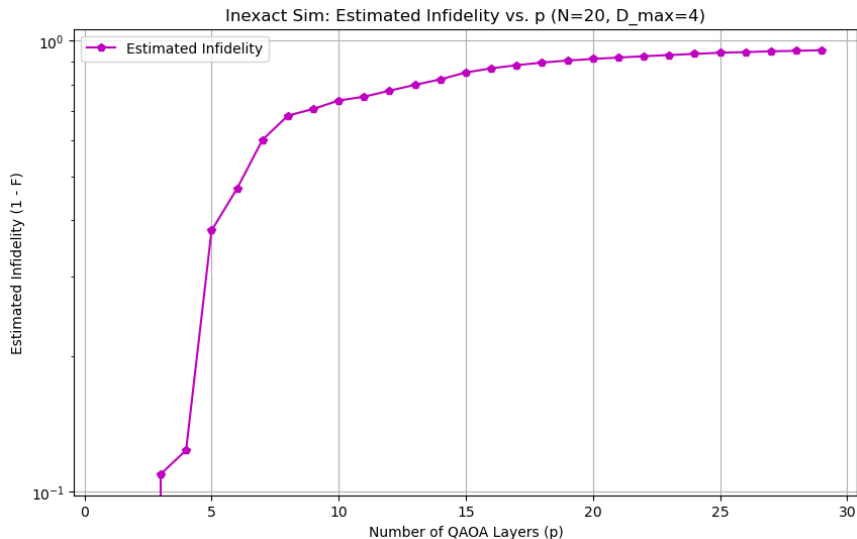


Figure 35: Estimated Infidelity vs. p ($N=20$).

Section 5

Conclusion & Future Outlook

Summary of Work Done

- Gained a comprehensive theoretical understanding of:
 - Matrix Product States (MPS) and their properties (canonical forms, entanglement).
 - Tensor decomposition methods (SVD, QR, RQ) and their physical significance.
 - Quantum Approximate Optimization Algorithm (QAOA), its origins, and problem mapping.
- Developed a robust Python framework for classical simulation of quantum circuits using MPS:
 - Implemented core MPS functionalities (canonicalization, inner product, gate application).
 - Included both exact and inexact (SVD & Variational) compression strategies.
- Established a foundation for analyzing QAOA performance:
 - Prepared to study entanglement growth.
 - Ready to evaluate approximation accuracy via fidelity estimation.

Key Principles Revisited

- **Entanglement as Resource and Bottleneck:**

- Quantum algorithms exploit entanglement for power.
- High entanglement leads to exponential bond dimension, bottlenecking classical simulation.

- **Approximation is Key:**

- Exact solutions for hard problems are intractable.
- Algorithms like QAOA/MPS compression provide "good enough" solutions within limited resources.

- **Hybrid Quantum-Classical Paradigm:**

- Quantum computers perform specific tasks (superpositions).
- Classical computers handle optimization, control, and data.
- Synergy enables practical near-term quantum algorithms.

- **Advanced QAOA Applications:**

- Simulate QAOA on more complex graph structures and optimization problems.
- Investigate the impact of different mixer Hamiltonians on performance.

- **Performance Optimization:**

- Optimize Python code for speed (e.g., JIT compilation, GPU acceleration).
- Integrate with specialized tensor network libraries for higher performance.

Section 6

References

Key Literature and Resources

- Ayral, T., Louvet, T., Zhou, Y., Lambert, C., Stoudenmire, E. M., Waintal, X. (2023). *Density-Matrix Renormalization Group Algorithm for Simulating Quantum Circuits with a Finite Fidelity*. PRX Quantum, 4(2), 020304.
- Dupont, M., Didier, N., Hodson, M. J., Moore, J. E., Reagor, M. J. (2022). *Entanglement perspective on the quantum approximate optimization algorithm*. Physical Review A, 106(2), 022423.
- Dubey, A., Zeybek, Z., Schmelcher, P. (2025). *Simulating Quantum Circuits with Tree Tensor Networks using Density-Matrix Renormalization Group Algorithm*. arXiv:2504.16718v1.
- Schollwöck, U. (2011). *The density-matrix renormalization group in the age of matrix product states*. Annals of Physics, 326(1), 96-192.
- Zhou, Y., Stoudenmire, E. M., Waintal, X. (2020). *What Limits the Simulation of Quantum Computers?*. Physical Review X, 10(4), 041038.

General Resources:

- TensorNetwork.org: Comprehensive resource for tensor network information.
- Tensors.net: General introduction to tensor networks.

Acknowledgements

- **Supervisor:** Prof. Dr. Peter Schmelcher (Universität Hamburg)
- **Guide:** Mr Zeki Zeybek (PhD Student, Universität Hamburg)
- **Local Guide:** Prof. Arvind (IISER Mohali)
- **Host Institution:** Center for Optical Quantum Technologies, Universität Hamburg, Germany
- **Home Institution:** Indian Institute of Science Education and Research (IISER), Mohali, India

Thank You!