Comprehensive Notes: Evolution of Matrix Product States (MPS) and 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

Document created during a research internship (May-July 2025)
at the Center for Optical Quantum Technologies,
University of Hamburg, Hamburg, Germany

Contents

I Time Evolution with Matrix Product States (MPS) 4

1 The Time-Evolving Block Decimation (TEBD) Algorithm 4
1.1 Matrix Product States (MPS) . . . 4
1.2 Trotter-Suzuki Decomposition . . . 4
1.3 Imaginary Time Evolution for Ground States . . . 5

2 Applying Gates to a Matrix Product State 5
2.1 Single-Qubit Gates . . . 5
2.2 Two-Qubit Gates and the SVD Truncation . . . 6

II The Quantum Approximate Optimization Algorithm (QAOA) 7

3 From Adiabatic Evolution to a Variational Algorithm 7
3.1 The Adiabatic Quantum Computation (AQC) Dream . . . 7
3.2 The QAOA Solution: Discretize and Train . . . 7

4 The Mechanics of a QAOA Layer: A Concrete Example 8
4.1 The Setup . . . 8
4.2 Step A: The Cost Layer — Phase Separation . . . 8
4.3 Step B: The Mixer Layer — Interference and Probability Flow . . . 8

5 The Complete Algorithm: Structure and Execution 9
5.1 The QAOA Ansatz and Circuit Structure . . . 9
5.2 Phase 1: The Optimization Loop . . . 9
5.3 Phase 2: The Final Measurement Run . . . 9

III Problem Formulation and Mapping for QAOA 10

6 Translating Classical Problems into QUBO 10
6.1 What is QUBO? . . . 10
6.2 Example Derivation: The Max-Cut Problem . . . 10

7 From QUBO to Quantum: The Ising Hamiltonian 11
7.1 The Mathematical Mapping . . . 11

8 Beyond QUBO: Diagonal Hamiltonians in Computational Basis 11
8.1 The "Rule": Why QUBO is the Standard Path . . . 12
8.2 The Deeper Truth: What QAOA *Actually* Needs . . . 12
8.3 The Exceptions: Beyond QUBO . . . 12
8.4 General Form of a Diagonal Hamiltonian in the Computational Basis . . . 12
8.4.1 Why This Form? The Operator Argument . . . 13
8.5 Summary Table . . . 14

9 Practical Representation: Sparse Pauli Operators 14

IV Advanced and Practical Implementations 15

10 Recursive QAOA (R-QAOA): A Divide and Conquer Heuristic 15
10.1 The R-QAOA Analogy: Solving a Sudoku Puzzle . . . 15
10.2 The Recursive Elimination Workflow . . . 15
10.3 Advantages and Disadvantages . . . 16

V Simulation, Execution, and Interpretation **17**

11 Advanced Simulation: Using Tensor Networks for R-QAOA **17**

12 Hardware vs. Simulation: The Role of Measurement **17**

13 Interpreting QAOA Results **18**
13.1 Degenerate Ground States and Multiple Solutions 18
13.2 Distinction from Grover's Search 19

VI Inexact Simulation of Quantum Circuits via a DMRG-like Algorithm **19**

14 Motivation: The Challenge of Exact Simulation **19**

15 The Algorithm: A DMRG-style Compression Step **20**

16 Example Comparison: Exact vs. Inexact Simulation **21**
16.1 Exact Simulation: 21
16.2 Inexact Simulation ($\chi_{\max} = 2$, $K = 1$ layer per compression): 22

17 Fidelity Estimation without the Exact State **22**
17.1 The Multiplicative Fidelity Law: 23
17.2 Detailed Mathematical Derivation: 23
17.3 The Physical Justification for the Approximation: 23

VII Summary and Conclusion **25**

18 Overview of the Journey **25**

19 Key Principles Revisited **25**

Part I
Time Evolution with Matrix Product States (MPS)

1 The Time-Evolving Block Decimation (TEBD) Algorithm

Simulating the full time evolution of a quantum many-body system is a formidable challenge. The Hilbert space of an N-qubit system grows as 2^N, making the storage of the full state vector impossible for even moderately sized systems on classical computers. Tensor network methods provide a powerful framework for efficiently representing and manipulating certain classes of quantum states, particularly those that obey an "area law" for entanglement, such as the ground states of local gapped Hamiltonians in one dimension.

Key Question:

What is the TEBD algorithm for MPS?

The Time-Evolving Block Decimation (TEBD) algorithm, developed by Guifré Vidal, is a highly effective numerical method for simulating the time evolution of 1D quantum many-body systems. Its core idea is to represent the quantum state as a **Matrix Product State (MPS)** and then evolve this state over small time steps by applying local quantum gates. These gates are derived from a **Trotter-Suzuki decomposition** of the global time evolution operator. The key to its efficiency lies in a controlled truncation step after each gate application that keeps the entanglement, and thus the computational cost, manageable.

1.1 Matrix Product States (MPS)

An MPS approximates the complex, high-dimensional state vector of a quantum system as a product of smaller tensors (often matrices), one for each site in the 1D chain. For an N-qubit system, a state $|\psi\rangle$ can be written as:

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

In an MPS representation, the massive coefficient tensor C is decomposed into a chain of smaller tensors:

$$C_{\sigma_1,\ldots,\sigma_N} \approx A^{\sigma_1}_{\alpha_0,\alpha_1} A^{\sigma_2}_{\alpha_1,\alpha_2} \cdots A^{\sigma_N}_{\alpha_{N-1},\alpha_N} \tag{2}$$

where the boundary indices α_0 and α_N are trivial (dimension 1). The maximum dimension of the virtual indices α_i is called the **bond dimension**, χ. This parameter controls the accuracy of the MPS approximation.

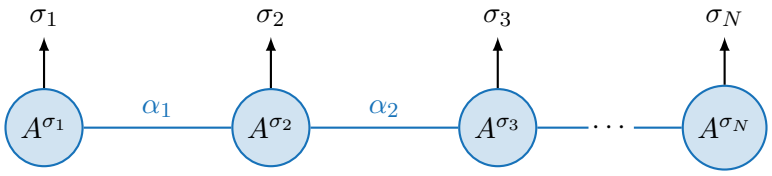


Figure 1: Diagram of a Matrix Product State (MPS). Each circle is a tensor A^{σ_i} with one physical index σ_i (black arrow) and two virtual or bond indices α_{i-1}, α_i (blue lines) that connect it to its neighbors.

1.2 Trotter-Suzuki Decomposition

To simulate time evolution $U(t) = e^{-iHt}$, we break down the Hamiltonian H into simpler, non-commuting parts. For a typical 1D system with only nearest-neighbor interactions, we can write $H = H_{\text{even}} + H_{\text{odd}}$, where:

- $H_{\text{even}} = \sum_{j=\text{even}} h_{j,j+1}$ contains all interactions on even bonds.
- $H_{\text{odd}} = \sum_{j=\text{odd}} h_{j,j+1}$ contains all interactions on odd bonds.

All terms within H_{even} commute with each other, as do all terms within H_{odd}. The evolution operator $e^{-iH\delta t}$ for a small time step δt can then be approximated.

First-Order Trotter Decomposition:

$$e^{-iH\delta t} = e^{-i(H_{\text{even}}+H_{\text{odd}})\delta t} \approx e^{-iH_{\text{even}}\delta t}e^{-iH_{\text{odd}}\delta t} + \mathcal{O}(\delta t^2) \tag{3}$$

This is the simplest but least accurate approximation.

Second-Order Trotter Decomposition (Symmetric): A more accurate and commonly used decomposition is:

$$e^{-iH\delta t} \approx e^{-iH_{\text{odd}}\frac{\delta t}{2}}e^{-iH_{\text{even}}\delta t}e^{-iH_{\text{odd}}\frac{\delta t}{2}} + \mathcal{O}(\delta t^3) \tag{4}$$

This symmetric form cancels the leading error term, making the simulation much more accurate for the same step size. Each of the exponentials on the right-hand side is now a product of commuting two-site gate operators, e.g., $e^{-iH_{\text{even}}\delta t} = \prod_{j=\text{even}} e^{-ih_{j,j+1}\delta t}$.

1.3 Imaginary Time Evolution for Ground States

A powerful application of TEBD is finding the ground state of a Hamiltonian. This is achieved by performing the evolution in **imaginary time**, $t \to -i\tau$. The evolution operator becomes $U(\tau) = e^{-H\tau}$.

$$|\psi(\tau)\rangle = e^{-H\tau}|\psi(0)\rangle = \sum_n e^{-E_n\tau}\langle E_n|\psi(0)\rangle |E_n\rangle \tag{5}$$

As $\tau \to \infty$, the term with the lowest energy E_0 (the ground state) will have the slowest exponential decay and will come to dominate the sum, provided the initial state $|\psi(0)\rangle$ has a non-zero overlap with it. The TEBD algorithm proceeds exactly as before, but with non-unitary gates $e^{-h_{j,j+1}\delta\tau}$, and the state must be re-normalized after each step.

2 Applying Gates to a Matrix Product State

The core operational primitive of any MPS-based algorithm is the application of quantum gates. This process involves local tensor contractions followed by a decomposition that restores the MPS structure.

2.1 Single-Qubit Gates

Applying a single-qubit gate U to site i is computationally simple as it does not change the entanglement structure of the state.

Mathematical Formulation

Let the MPS tensor at site i be $A^{\sigma_i}_{\alpha_{i-1},\alpha_i}$. The gate U has matrix elements U_{σ_i',σ_i}. The new tensor A' is formed by contracting the gate with the physical index of the original tensor:

$$(A')^{\sigma_i'}_{\alpha_{i-1},\alpha_i} = \sum_{\sigma_i} U_{\sigma_i',\sigma_i} A^{\sigma_i}_{\alpha_{i-1},\alpha_i} \tag{6}$$

Crucially, the bond dimension χ does not change. The cost is $\mathcal{O}(d^2\chi^2)$, where d is the physical dimension.

Diagrammatic Representation

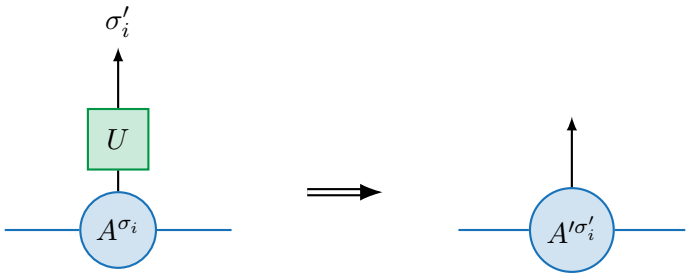


Figure 2: Application of a single-qubit gate U to an MPS tensor A. The gate is simply contracted with the physical leg, producing a new tensor A' of the same shape.

2.2 Two-Qubit Gates and the SVD Truncation

Applying a two-qubit gate U to neighboring sites i and $i+1$ is the essential step of TEBD. This operation can increase the entanglement between the two sites, and we must use a controlled approximation to prevent the bond dimension from growing indefinitely.

Mathematical and Diagrammatic Workflow

The process involves four key steps for each bond:

1. **Contraction:** The two tensors A^{σ_i} and $A^{\sigma_{i+1}}$ are contracted over their shared virtual index α_i to form a single, larger four-index tensor Θ.

$$\Theta^{\sigma_i,\sigma_{i+1}}_{\alpha_{i-1},\alpha_{i+1}} = \sum_{\alpha_i} A^{\sigma_i}_{\alpha_{i-1},\alpha_i} A^{\sigma_{i+1}}_{\alpha_i,\alpha_{i+1}} \tag{7}$$

2. **Gate Application:** The two-site unitary gate U is applied to the physical indices of this combined tensor Θ to form Θ'.

$$(\Theta')^{\sigma'_i,\sigma'_{i+1}}_{\alpha_{i-1},\alpha_{i+1}} = \sum_{\sigma_i,\sigma_{i+1}} U^{\sigma'_i,\sigma'_{i+1}}_{\sigma_i,\sigma_{i+1}} \Theta^{\sigma_i,\sigma_{i+1}}_{\alpha_{i-1},\alpha_{i+1}} \tag{8}$$

3. **Decomposition (SVD):** The resulting tensor Θ' is reshaped into a matrix M by grouping indices: $M_{(\sigma'_i,\alpha_{i-1}),(\sigma'_{i+1},\alpha_{i+1})}$. A Singular Value Decomposition (SVD) is performed on this matrix: $M = U'SV^\dagger$. The diagonal matrix S contains the singular values λ_α, which are the Schmidt coefficients across the bond $(i, i+1)$.

4. **Truncation:** The diagonal matrix of singular values, S, is truncated by keeping only the χ largest values, creating a new matrix S_χ. This is the "block decimation" step. The truncation error is the sum of the squares of the discarded singular values: $\epsilon = \sum_{\alpha=\chi+1}^{d\chi} \lambda_\alpha^2$. The new MPS tensors are then formed:

$$(A')^{\sigma'_i}_{\alpha_{i-1},\alpha'_i} = U'_{(\sigma'_i,\alpha_{i-1}),\alpha'_i} \tag{9}$$

$$(B')^{\sigma'_{i+1}}_{\alpha'_i,\alpha_{i+1}} = (S_\chi V^\dagger)_{\alpha'_i,(\sigma'_{i+1},\alpha_{i+1})} \tag{10}$$

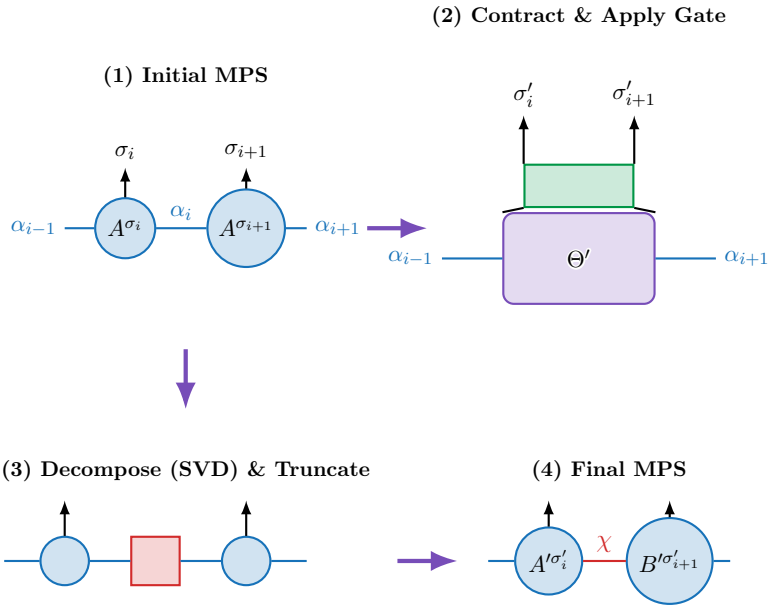


Figure 3: The TEBD update step for a two-qubit gate. The new bond dimension is truncated to χ (red line), which controls the computational cost and accuracy of the simulation.

Part II
The Quantum Approximate Optimization Algorithm (QAOA)

3 From Adiabatic Evolution to a Variational Algorithm

The motivation for QAOA comes from a beautiful but often impractical method of quantum computation known as Adiabatic Quantum Computation (AQC). Understanding this origin helps clarify why QAOA is structured the way it is.

3.1 The Adiabatic Quantum Computation (AQC) Dream

The AQC approach proposes to solve an optimization problem by:

1. Encoding the problem solution into the ground state of a complex **Problem Hamiltonian**, H_C.
2. Preparing a quantum system in the simple, easy-to-prepare ground state of a **Beginning Hamiltonian**, H_B.
3. Slowly and continuously transforming the system's Hamiltonian from H_B to H_C over a long time T.

The **Adiabatic Theorem** of quantum mechanics guarantees that if this transformation is done "slowly enough," the system will remain in the ground state throughout the evolution. At the end, measuring the system reveals the ground state of H_C, which is the desired solution. The catch is that for many hard problems, the required evolution time T becomes exponentially long, making the protocol infeasible on real quantum hardware which suffers from decoherence.

3.2 The QAOA Solution: Discretize and Train

QAOA can be seen as a practical, "digitized" version of AQC. Instead of a slow, continuous evolution, QAOA approximates it with a series of discrete steps, making it suitable for gate-based quantum computers.

1. **Discretization (Trotterization):** The smooth evolution is replaced by alternating applications of the two Hamiltonians for short periods: first evolving under H_C for a duration γ, then under H_B for a duration β.
2. **Variational Parameters:** Instead of relying on a pre-determined slow schedule, QAOA treats the durations (γ, β) as **trainable parameters**. A classical computer's job is to find the optimal values of these parameters to best guide the quantum state towards the true ground state.

This transforms the problem from one of physical simulation into one of hybrid quantum-classical optimization.

Key Question:

If we build the cost Hamiltonian for a problem, why not just find its ground state directly? Why do we need these complex algorithms?

This is the central challenge. For a system of N qubits, the Hamiltonian is a $2^N \times 2^N$ matrix.

- **Classical Impracticality:** Storing and diagonalizing this matrix on a classical computer becomes impossible for $N > 30 - 40$. The memory and time required scale exponentially.
- **Quantum Limitation:** Even a quantum computer cannot "look at" a Hamiltonian and instantly find its ground state. The laws of quantum mechanics (specifically, the nature of measurement and the unitarity of operations) forbid such a direct "searchlight." Finding the ground state of an arbitrary local Hamiltonian is a QMA-complete problem, believed to be hard even for quantum computers.

Therefore, algorithms like QAOA are not direct solvers; they are sophisticated **strategies** or heuristics designed to prepare a quantum state that has a very high probability of being the true ground state.

4 The Mechanics of a QAOA Layer: A Concrete Example

To understand how QAOA works, we must move beyond analogy and look at the quantum mechanics of a single layer. We will use the simplest non-trivial example: a 2-qubit Max-Cut problem for two connected nodes.

4.1 The Setup

- **Problem:** Nodes 0 and 1 are connected. The optimal solutions place them in different sets, corresponding to the quantum states $|01\rangle$ and $|10\rangle$.
- **Cost Hamiltonian H_C:** We use $H_C = Z_0 Z_1$. This operator assigns a low energy (-1) to the good states $|01\rangle, |10\rangle$ and a high energy (+1) to the bad states $|00\rangle, |11\rangle$.
- **Mixer Hamiltonian H_B:** The standard mixer is $H_B = X_0 + X_1$.
- **Initial State:** We begin in a uniform superposition of all four states:

$$|\psi_0\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \tag{11}$$

4.2 Step A: The Cost Layer — Phase Separation

The first step in a QAOA layer is to apply the Cost evolution $U_C(\gamma) = e^{-i\gamma H_C}$. Because H_C is diagonal in the computational basis, this operator **only changes the phase** of each basis state, it does not change their probability amplitudes. The applied phase is proportional to the state's energy (cost).

Applying this to our initial state:

$$\begin{aligned}
|\psi_1\rangle &= e^{-i\gamma Z_0 Z_1}|\psi_0\rangle \\
&= \frac{1}{2}e^{-i\gamma(1)}|00\rangle + \frac{1}{2}e^{-i\gamma(-1)}|01\rangle + \frac{1}{2}e^{-i\gamma(-1)}|10\rangle + \frac{1}{2}e^{-i\gamma(1)}|11\rangle \\
&= \frac{1}{2}\left(e^{-i\gamma}|00\rangle + e^{+i\gamma}|01\rangle + e^{+i\gamma}|10\rangle + e^{-i\gamma}|11\rangle\right)
\end{aligned} \tag{12}$$

The result: We have "marked" the good and bad solutions. The good states ($|01\rangle, |10\rangle$) now have a phase of $+e^{i\gamma}$, while the bad states ($|00\rangle, |11\rangle$) have a phase of $+e^{-i\gamma}$. The probability of measuring any state is still 25%, but the states are now distinguishable by their phase.

4.3 Step B: The Mixer Layer — Interference and Probability Flow

The second step applies the Mixer evolution $U_B(\beta) = e^{-i\beta H_B}$. The Mixer Hamiltonian H_B is non-diagonal and its role is to make the amplitudes of the basis states "interfere" with one another. It uses the phase differences created by the Cost Layer to control this interference.

The Mixer causes the amplitude of each state to "spill over" to its neighbors (states that are one bit-flip away).

- **Constructive Interference:** Amplitude flowing from bad states (phase $e^{-i\gamma}$) towards good states (phase $e^{+i\gamma}$) can be made to add up.
- **Destructive Interference:** Amplitude flowing between states with different phases can be made to cancel out.

For the right choice of β, the net effect is a flow of probability amplitude away from the high-cost states and towards the low-cost states. After this layer, the state becomes:

$$|\psi_2\rangle = \mathcal{A}|00\rangle + \mathcal{B}|01\rangle + \mathcal{B}|10\rangle + \mathcal{A}|11\rangle \tag{13}$$

where, for optimal angles, the magnitude of the "good" amplitude $\mathcal{B}$ will be larger than the magnitude of the "bad" amplitude $\mathcal{A}$. The probability of measuring the correct answers, $|\mathcal{B}|^2$, has increased.

Key Question:

What if the classical optimization is wrong? Can the amplitude for the bad state increase?

Yes, absolutely. The mixer layer is a tool, and the angles (γ, β) are the instructions. Incorrect angles will cause the interference to work against you. A poor choice of β could cause destructive interference at the good states and constructive interference at the bad states, increasing the probability of measuring an incorrect answer. This is precisely why a "smart" classical optimizer is required to find the angles that harness the interference for our benefit.

5 The Complete Algorithm: Structure and Execution

5.1 The QAOA Ansatz and Circuit Structure

The full QAOA algorithm prepares a state $|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$ by repeating the two-step process described above for p layers. The integer p is the depth of the algorithm.

$$|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \left[\prod_{k=1}^{p} e^{-i\beta_k H_B} e^{-i\gamma_k H_C}\right] |+\rangle^{\otimes N} \tag{14}$$

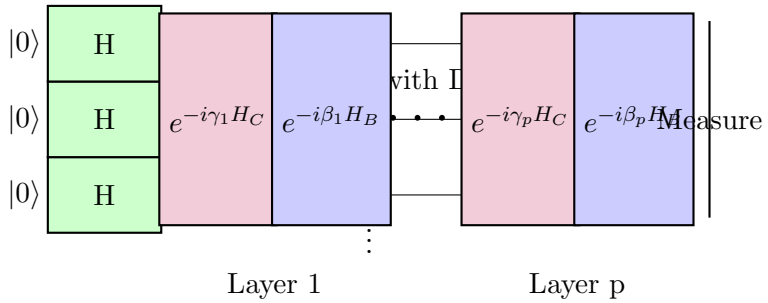


Figure 4: The general structure of a QAOA circuit. After initialization, p layers of alternating Cost and Mixer Hamiltonian evolutions are applied. The durations (γ_k, β_k) are the $2p$ parameters to be optimized.

5.2 Phase 1: The Optimization Loop

The process of finding the optimal angles $(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)$ is a hybrid loop.

1. **Suggest:** A classical optimizer proposes a set of $2p$ angles.
2. **Execute & Score:** The quantum computer builds the p-layered circuit with these angles. It then runs this circuit many times (thousands of "shots") to build up statistics and compute a reliable estimate of the average energy, $E = \langle H_C \rangle$. This score is returned.
3. **Improve:** The optimizer assesses the score and uses a classical algorithm (e.g., COBYLA) to choose a new set of angles for the next iteration, aiming to lower the energy.

This loop continues until the energy score converges to a minimum.

5.3 Phase 2: The Final Measurement Run

Once the optimization loop has found the best angles, the goal changes from finding energy to finding the solution state.

1. Build the QAOA circuit one last time using the optimal angles.
2. Run this circuit thousands of times.
3. For each run, measure the final state in the computational basis to get a classical bitstring (e.g., '1001').
4. Tally the results in a histogram. The bitstring that appears most frequently is the algorithm's approximate solution to the optimization problem. If multiple strings have a high frequency, it indicates multiple optimal solutions exist.

Part III
Problem Formulation and Mapping for QAOA

6 Translating Classical Problems into QUBO

Before a quantum algorithm like QAOA can be applied, the real-world optimization problem must be translated into a standardized mathematical format that can be mapped to a quantum system. The most common format for this is the **QUBO** model.

6.1 What is QUBO?

QUBO stands for **Q**uadratic **U**nconstrained **B**inary **O**ptimization. It's a standard format for a wide range of combinatorial optimization problems.

- **Binary:** The decision variables of the problem can only take two values, typically $\{0, 1\}$.
- **Unconstrained:** There are no explicit constraints on the variables (e.g., "the sum of variables must be less than 5"). All the problem's logic, including any penalties for breaking constraints, must be incorporated into a single cost function.
- **Quadratic:** The cost function to be minimized is a polynomial of at most degree two.

Any QUBO problem can be expressed in a universal matrix form:

$$\min_{x \in \{0,1\}^N} C(x) = x^T Q x \tag{15}$$

where x is the column vector of N binary variables, and Q is an $N \times N$ matrix of real-valued coefficients that defines the specific problem. The diagonal elements Q_{ii} encode the linear costs associated with each variable, and the off-diagonal elements Q_{ij} encode the quadratic costs of variable interactions.

6.2 Example Derivation: The Max-Cut Problem

Let's derive the QUBO formulation for the Max-Cut problem step-by-step.

- **Goal:** Partition the vertices of a graph $G = (V, E)$ into two sets, which we'll label 0 and 1, to maximize the number of edges that connect vertices in different sets.
- **Variables:** Assign a binary variable $x_i \in \{0, 1\}$ to each vertex $i \in V$.
- **Cost Function Logic:** We want to build a function that "rewards" us for each cut edge. For a single edge $(i, j) \in E$, it is cut if and only if $x_i \neq x_j$. A simple algebraic expression for this is $(x_i - x_j)^2$.
  - If $x_i = x_j$ (not cut), then $(x_i - x_j)^2 = 0$.
  - If $x_i \neq x_j$ (is cut), then $(1 - 0)^2 = 1$ or $(0 - 1)^2 = 1$.

Expanding this term gives: $(x_i - x_j)^2 = x_i^2 - 2x_i x_j + x_j^2$. Since our variables are binary, we can use the crucial identity $x_i^2 = x_i$. This simplifies the expression for a single cut edge to $x_i + x_j - 2x_i x_j$.

The total number of cuts is the sum of this expression over all edges in the graph:

$$\text{Cuts}(x) = \sum_{(i,j) \in E} (x_i + x_j - 2x_i x_j) \tag{16}$$

Our goal is to maximize this function. Standard optimizers, however, are typically designed to *minimize* a function. Maximizing a function is equivalent to minimizing its negative. Therefore, the cost function $C(x)$ for Max-Cut is:

$$C(x)_{\text{Max-Cut}} = \sum_{(i,j) \in E} (2x_i x_j - x_i - x_j) \tag{17}$$

This is a quadratic polynomial of binary variables, so it is in QUBO form. We can now construct the Q matrix by matching the coefficients of this sum to the terms in $x^T Q x$.

7 From QUBO to Quantum: The Ising Hamiltonian

Key Question:

What is a cost function Hamiltonian and where does it come from?

A Cost Hamiltonian H_C is a quantum operator constructed such that its energy spectrum perfectly mirrors the values of a classical cost function. For any possible solution (represented by a computational basis state), the energy (eigenvalue) of that state under H_C is equal to the classical cost of that solution. This allows us to rephrase an abstract optimization problem as a physical problem of finding a system's lowest energy state.

The bridge from the classical QUBO model to a quantum Hamiltonian is the realization that QUBO problems are mathematically equivalent to finding the ground state of an **Ising Model**, a cornerstone of statistical physics.

7.1 The Mathematical Mapping

To create this equivalence, we need a way to map classical binary variables to quantum operators. The standard mapping uses the eigenvalues of the Pauli-Z operator.

- A classical variable $x_i \in \{0, 1\}$.
- A qubit state in the Z-basis is either $|0\rangle$ (eigenvalue +1) or $|1\rangle$ (eigenvalue -1).

We can establish a direct mapping between them with the following transformation:

$$x_i \longleftrightarrow \frac{I - Z_i}{2} \tag{18}$$

where Z_i is the Pauli-Z operator acting on qubit i and I is the identity.

- If qubit i is in state $|0\rangle$ (eigenvalue +1), the mapping gives $x_i = (1 - (+1))/2 = 0$.
- If qubit i is in state $|1\rangle$ (eigenvalue -1), the mapping gives $x_i = (1 - (-1))/2 = 1$.

By substituting this operator into our QUBO cost function for every variable x_i, the classical function is transformed into a quantum Hamiltonian.

- A linear term $c_i x_i$ becomes a term acting on a single qubit, $\propto Z_i$. This is known as a "local field" term.
- A quadratic term $c_{ij} x_i x_j$ becomes a term acting on two qubits, $\propto Z_i Z_j$. This is a two-body "interaction" term.

After performing this substitution and collecting terms (while ignoring constant energy shifts that don't affect the ground state), the resulting operator will always have the form of the Ising Hamiltonian:

$$H_C = \sum_i h_i Z_i + \sum_{i<j} J_{ij} Z_i Z_j \tag{19}$$

This Hamiltonian is diagonal in the computational basis, and its eigenvalues are the costs of the corresponding classical solutions, fulfilling our requirement.

8 Beyond QUBO: Diagonal Hamiltonians in Computational Basis

Key Question:

Are all QAOA algorithms for all sort of optimisation problems based on QUBO? So only if the optimisation problem can be mapped to a QUBO, QAOA can be applied?

This is an absolutely crucial question that gets to the scope and limitations of QAOA. The short answer is: No, a problem doesn't *strictly* have to be mapped to a QUBO. However, QUBO is the most common and straightforward path because it naturally leads to the kind of Hamiltonian that QAOA requires.

Let's break this down into the **rule** and the **exceptions**.

8.1 The "Rule": Why QUBO is the Standard Path

For the vast majority of common combinatorial optimization problems, the standard pipeline is:

$$\text{Problem} \rightarrow \text{QUBO} \rightarrow \text{Ising Hamiltonian} \rightarrow \text{QAOA}$$

This works perfectly for a huge class of important problems such as Max-Cut, Minimum Vertex Cover, and the Traveling Salesman Problem (TSP).
Why is this the standard?

- **Universality:** QUBO is a powerful and universal format. An enormous number of NP-hard problems can be formulated as a QUBO, making it the "lingua franca" for this type of optimization.
- **Simplicity:** The mapping from a QUBO model to a 2-local Ising Hamiltonian (of the form $\sum h_i Z_i + \sum J_{ij} Z_i Z_j$) is a fixed, mechanical recipe that is reliable and easy to automate.
- **Hardware Compatibility:** The resulting 2-local Ising Hamiltonian is the "native" language of many quantum devices and simulators. The ZZ interactions are often easier to implement than more complex interactions.

So, for most practical purposes, if you can get your problem into QUBO form, you are guaranteed to be able to apply QAOA.

8.2 The Deeper Truth: What QAOA *Actually* Needs

The strict requirement for QAOA is not a QUBO model. The fundamental requirement is the ability to construct a **Cost Hamiltonian (H_C)** that has one critical property:

$$H_C \textbf{ must be diagonal in the computational basis.}$$

This means that the basis states (the binary strings like $|01101\ldots\rangle$) are the eigenstates of H_C. When H_C acts on one of these states, it just multiplies it by a number which represents the energy or cost.

$$H_C|s\rangle = \text{Cost}(s)|s\rangle$$

An Ising Hamiltonian is the classic example of a diagonal Hamiltonian, but others exist, which leads us to the exceptions.

8.3 The Exceptions: Beyond QUBO

1. Higher-Order Optimization (PUBO Problems) What if your cost function involves three or more variables multiplied together, such as $x_i x_j x_k$? This is no longer a QUBO problem, but a **PUBO** (Polynomial Unconstrained Binary Optimization) problem. QAOA can still solve this by mapping the problem directly to a **k-local Hamiltonian** with terms like $Z_i Z_j Z_k$. This Hamiltonian is still diagonal, so QAOA's structure remains unchanged, even if the circuit implementation is more complex.

2. Constrained Optimization Problems Many real-world problems have constraints, like the Knapsack Problem. QAOA handles these using **penalty terms**. You formulate two Hamiltonians: one for the objective ($H_{\text{objective}}$) and one for the constraints ($H_{\text{constraint}}$), which assigns a high energy penalty to invalid solutions. The final Cost Hamiltonian is a sum:

$$H_C = H_{\text{objective}} + P \cdot H_{\text{constraint}}$$

Here, P is a large penalty coefficient. As long as both components are diagonal, their sum H_C is also diagonal, and QAOA can be applied.

8.4 General Form of a Diagonal Hamiltonian in the Computational Basis

Question: *Can you tell me what a general hamiltonian for N particles would look like for it be diagonal in computational basis?*

Answer: A Hamiltonian for N particles (qubits) is diagonal in the computational basis if and only if it can be written exclusively using Pauli-Z operators and the Identity (I) operator. It cannot contain any Pauli-X or Pauli-Y operators, as these would create off-diagonal terms by changing the basis states (e.g., $X|0\rangle = |1\rangle$).

The General Form

The most general form of an N-particle Hamiltonian that is diagonal in the computational basis is an expansion of products of Z operators, often called a **k-local Ising Hamiltonian**:

$$H_C = c_0 I + \sum_i h_i Z_i + \sum_{i<j} J_{ij} Z_i Z_j + \sum_{i<j<k} K_{ijk} Z_i Z_j Z_k + \dots$$

Let's break down what each part means:

- $c_0 I$: A constant energy offset that applies to all states equally. This term doesn't affect which state has the lowest energy.
- $\sum h_i Z_i$: These are the **1-local terms**, also known as linear terms or "biases". The coefficient h_i represents the strength of a local field on qubit i.
- $\sum J_{ij} Z_i Z_j$: These are the **2-local terms**, also known as quadratic terms or "couplings". The coefficient J_{ij} represents the interaction strength between qubit i and qubit j.
- $\sum K_{ijk} Z_i Z_j Z_k + \dots$: These are higher-order, **k-local terms** representing interactions between three or more qubits.

Most common optimization problems mapped from a QUBO model only require terms up to 2-local ($Z_i Z_j$).

8.4.1 Why This Form? The Operator Argument

A Hamiltonian H is diagonal in the computational basis if, when it acts on any basis state $|s\rangle$, the result is just a number (the energy E_s) times that same state: $H|s\rangle = E_s|s\rangle$.

- **Pauli-Z**: $Z|0\rangle = +1|0\rangle$ and $Z|1\rangle = -1|1\rangle$. The Z operator **preserves** the basis states. Therefore, any Hamiltonian made only of Z and I operators will be diagonal.
- **Pauli-X**: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. The X operator **flips** the basis state, creating off-diagonal elements in the Hamiltonian's matrix representation.
- **Pauli-Y**: Similarly, Y also flips states ($Y|0\rangle = i|1\rangle$) and is therefore also forbidden.

A Concrete 2-Qubit Example

Let's take $N = 2$ and a simple diagonal Hamiltonian:

$$H = 5Z_1 - 2Z_1 Z_2$$

This Hamiltonian is diagonal in the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. We can calculate the energy of each basis state by remembering that Z on $|0\rangle$ gives an eigenvalue of $+1$ and on $|1\rangle$ gives -1.

- **Energy of** $|00\rangle$: $H|00\rangle = (5(+1) - 2(+1)(+1))|00\rangle = +3|00\rangle \implies E = 3$.
- **Energy of** $|01\rangle$: $H|01\rangle = (5(+1) - 2(+1)(-1))|01\rangle = +7|01\rangle \implies E = 7$.
- **Energy of** $|10\rangle$: $H|10\rangle = (5(-1) - 2(-1)(+1))|10\rangle = -3|10\rangle \implies E = -3$.
- **Energy of** $|11\rangle$: $H|11\rangle = (5(-1) - 2(-1)(-1))|11\rangle = -7|11\rangle \implies E = -7$.

The lowest energy state (the ground state) is $|11\rangle$. If we were to write this H as a matrix in the ordered basis $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$, it would be perfectly diagonal:

$$H = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & -7 \end{pmatrix}$$

8.5 Summary Table

Problem Type	Can it be mapped to QUBO?	Can QAOA be applied?	How?
Standard Combinatorial (e.g., Max-Cut)	**Yes** (naturally)	**Yes**	Map to QUBO, then to a 2-local Ising Hamiltonian.
Higher-Order (e.g., Max-3-SAT)	No (it's a PUBO)	**Yes**	Map directly to a k-local Ising Hamiltonian (e.g., with $Z_iZ_jZ_k$ terms).
Constrained (e.g., Knapsack Problem)	Not directly	**Yes**	Use a penalty method. Create $H_C = H_{\text{objective}} + P \cdot H_{\text{constraint}}$.

Conclusion: Thinking in terms of QUBO is an extremely useful and powerful shortcut for applying QAOA. However, the fundamental requirement is more general: as long as you can encode your problem's cost function (including any constraints via penalties) into a **diagonal Hamiltonian**, you can unleash QAOA on it.

9 Practical Representation: Sparse Pauli Operators

Storing the $2^N \times 2^N$ matrix for H_C is impossible. Instead, we use a **sparse** representation, which is a highly efficient data structure for this task.

Key Question:

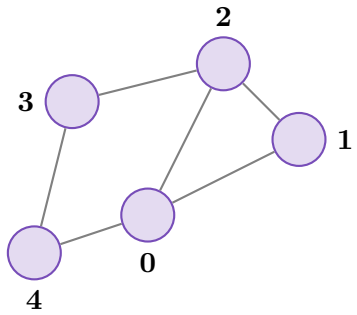
What are sparse Pauli operators?

A Sparse Pauli Operator is a data structure that represents a large operator as a compact "recipe" rather than an explicit matrix. It consists of two lists:

1. A list of **Pauli Strings** (e.g., ''ZIZII'').
2. A list of corresponding numerical **coefficients**.

This is possible because the Hamiltonians for most relevant problems are themselves sparse, meaning they are a sum of only a few simple terms, not a dense combination of all possible interactions.

For example, consider the Max-Cut problem for the 5-node graph below. The cost Hamiltonian is the sum of Z_iZ_j terms for each edge.



The Hamiltonian is $H_C = Z_0Z_1 + Z_0Z_2 + Z_0Z_4 + Z_1Z_2 + Z_2Z_3 + Z_3Z_4$. Using a standard convention where strings are read from left (qubit 4) to right (qubit 0), this is represented by software libraries as:

Cost Function Hamiltonian:

```
SparsePauliOp(
  paulis=['IIIZZ', 'IIZIZ', 'ZIIIZ', 'IIZZI', 'IZZII', 'ZZIII'],
  coeffs=[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
)
```

This compact, human-readable format is how quantum software defines and manipulates Hamiltonians for problems of any meaningful size, making the simulation and execution of QAOA computationally tractable.

Part IV
Advanced and Practical Implementations

10 Recursive QAOA (R-QAOA): A Divide and Conquer Heuristic

While standard QAOA is a powerful theoretical tool, it suffers from a major practical bottleneck: the classical optimization of the $2p$ variational parameters. As the problem size N and the circuit depth p increase, the optimization landscape becomes vast and complex, making it exponentially difficult for a classical optimizer to find the global minimum. The Recursive QAOA (R-QAOA) was developed to circumvent this challenge.

Key Question:

Can you explain recursive QAOA in detail and simply?

R-QAOA is a heuristic algorithm that reframes the optimization problem. Instead of trying to solve for all N variables at once in a single, deep QAOA circuit, R-QAOA solves the problem iteratively, one variable at a time. It uses a series of very "cheap," low-depth QAOA runs to make confident, greedy decisions, simplifying the problem at each step.

10.1 The R-QAOA Analogy: Solving a Sudoku Puzzle

Imagine solving a Sudoku puzzle. A naive approach would be to guess all 81 numbers at once and check if the board is valid—an impossible task. The human approach is recursive:

1. Scan the board to find a square where you are **most certain** of the number.
2. Confidently write that number in, permanently.
3. This action **simplifies** the rest of the puzzle by eliminating possibilities in the corresponding row, column, and box.
4. **Repeat** the process: find the next most certain number on the now-simpler board, and continue until the puzzle is solved.

R-QAOA applies this exact "divide and conquer" logic to optimization problems.

10.2 The Recursive Elimination Workflow

Starting with an N-variable problem, R-QAOA enters a loop that continues until all variables have been assigned a value. At each step k (from $k = N$ down to 1 variable):

1. **Run a "Shallow" QAOA:** Perform a standard QAOA run on the current, k-variable problem, but with a very small depth, almost always $\mathbf{p = 1}$. This is computationally cheap, as the classical optimizer only needs to find two angles (γ_1, β_1). We don't need a perfect answer, just a "good guess."
2. **Find the "Most Certain" Variable:** After the 'p=1' QAOA run, we measure the qubits many times to estimate the expectation value of the Pauli-Z operator for each remaining variable, $\langle Z_i \rangle$. This value tells us the variable's bias:
   - $\langle Z_i \rangle \approx +1 \implies$ Qubit i is very likely to be in state $|0\rangle$.
   - $\langle Z_i \rangle \approx -1 \implies$ Qubit i is very likely to be in state $|1\rangle$.
   - $\langle Z_i \rangle \approx 0 \implies$ The state of qubit i is highly uncertain.

   We identify the variable j for which the absolute value $|\langle Z_j \rangle|$ is the largest. This is our "most certain" variable, akin to the most obvious number in the Sudoku.
3. **Eliminate the Variable and Simplify:** We make a "greedy" decision to permanently fix the value of variable j. We set its classical value x_j according to the sign of its expectation value (e.g., $x_j = 1$ if $\langle Z_j \rangle < 0$). This variable is now removed from the problem.

4. **Update the Hamiltonian:** The cost Hamiltonian is updated to reflect this fixed value. For example, if we fix variable $x_j = -1$ (in spin form), a term in the Hamiltonian like $J_{ij}Z_iZ_j$ simplifies to $-J_{ij}Z_i$, reducing a two-body interaction to a one-body local field. The problem is now a simpler, $(k-1)$-variable optimization.

5. **Recurse:** The algorithm loops back to Step 1 with the new, smaller problem.

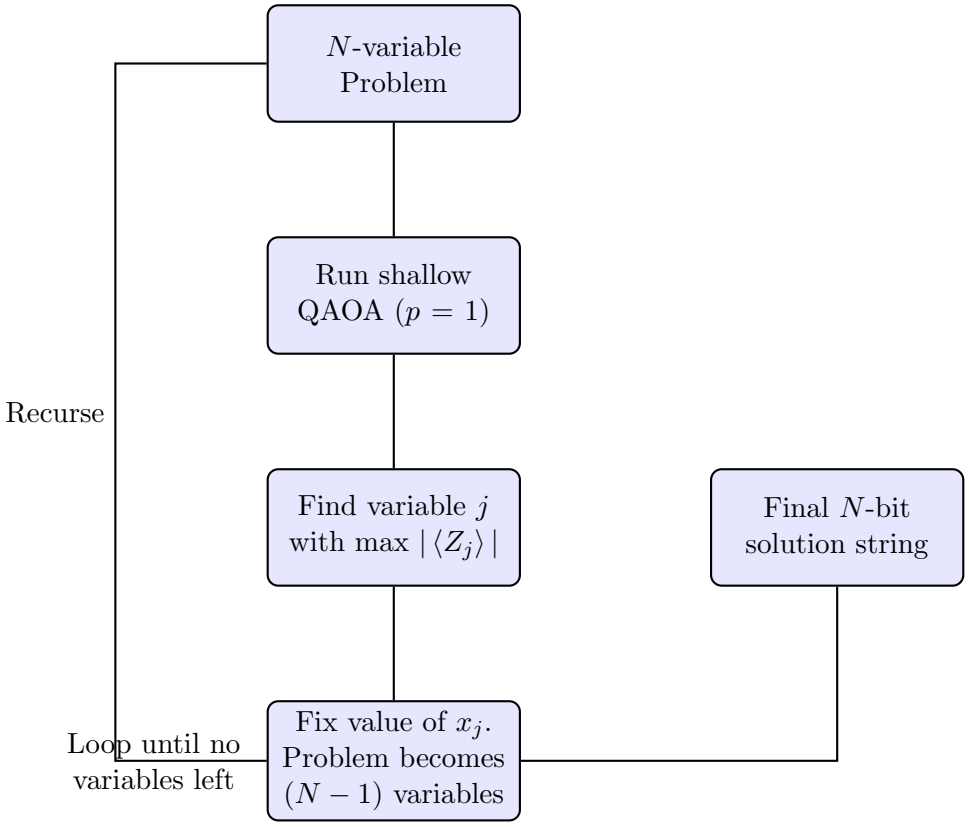


Figure 5: The iterative workflow of the Recursive QAOA algorithm. At each step, one variable is confidently eliminated, simplifying the problem for the next iteration.

10.3 Advantages and Disadvantages

R-QAOA provides a practical path towards solving larger optimization problems.

Advantages:

- **Avoids Hard Classical Optimization:** It completely sidesteps the difficult task of optimizing a large number of parameters in a high-dimensional space. It only ever solves the easy 'p=1' optimization problem.
- **Scalability:** It offers a more practical approach for near-term quantum devices, as the quantum resources required at each step (a shallow circuit on a decreasing number of qubits) are much more modest.
- **Performance:** For many problems, R-QAOA has been shown to achieve better approximation ratios than standard QAOA with a reasonable circuit depth p.

The Main Disadvantage:

- **Greedy Nature:** The algorithm is "greedy" because the decision made at each step is final and cannot be undone. If it makes an error by eliminating a variable to the wrong value early on, it can never recover from that mistake. Despite this, the heuristic has proven to be remarkably effective in practice.

Part V
Simulation, Execution, and Interpretation

11 Advanced Simulation: Using Tensor Networks for R-QAOA

While QAOA is designed as a hybrid quantum-classical algorithm, its "quantum" component can often be simulated effectively on a classical computer, especially for problems with limited entanglement structure. This leads to a powerful, purely classical approach using tensor networks.

Key Question:

Can we use tensor networks for R-QAOA?

Yes, absolutely. In this approach, the Matrix Product State (MPS) tensor network acts as a classical simulator for the quantum circuit. Instead of running on a QPU, all operations are performed on the MPS representation of the state.

This changes the R-QAOA loop significantly:

1. The initial state $|+\rangle^{\otimes N}$ is constructed as a simple MPS with bond dimension $\chi = 1$.
2. The Cost and Mixer evolution operators ($e^{-i\gamma H_C}, e^{-i\beta H_B}$) are applied directly to the MPS using the TEBD techniques described in Part I. The key difference is that the gates here are defined by the QAOA Hamiltonians, not a generic 1D model.
3. After the circuit is simulated, the expectation values $\langle Z_i \rangle$ needed for the recursive step are calculated **directly and exactly** from the final MPS. This completely avoids the need for statistical sampling ("shots").
4. The variable with the largest $|\langle Z_i \rangle|$ is eliminated, the Hamiltonian is simplified, and the process repeats on a smaller system.

Advantages of the Tensor Network Approach

- **No Quantum Hardware Needed:** The entire algorithm can be run on a classical computer, making it highly accessible.
- **Noise-Free:** The simulation is free from the physical decoherence that affects real QPUs, allowing for a clean study of the algorithm's performance.
- **Efficient and Exact Expectation Values:** Calculating $\langle Z_i \rangle$ is a deterministic operation on the MPS, which is much faster and more accurate than the thousands of shots required on hardware.

The Limitation: The Entanglement Bottleneck

The efficiency of an MPS simulation is limited by its bond dimension χ. The QAOA circuit, especially the Cost Layer for highly connected problems, generates entanglement across the system. If the entanglement grows too large, the required bond dimension to accurately represent the state grows exponentially, and the classical simulation becomes intractable. This approach works best for problems on graphs with low connectivity, like chains or sparse graphs.

12 Hardware vs. Simulation: The Role of Measurement

There is a fundamental difference between simulating QAOA and running it on real quantum hardware, which centers on how information is extracted.

Key Question:

In a simulation, do we need to run the circuit thousands of times (shots) to get the energy?

No. This is the crucial distinction.

- **On a Classical Simulator:** The computer maintains a complete mathematical description of the state at all times—the **state vector** (an array of 2^N complex numbers). After the circuit is simulated once, the simulator has the exact final state vector $|\psi_{final}\rangle$. It can then compute the energy directly and exactly using the mathematical formula $\langle E\rangle = \langle\psi_{final}|H_C|\psi_{final}\rangle$. No repetition is needed.
- **On Real Quantum Hardware:** The state vector is a hidden physical property of the qubits. We are forbidden by the laws of physics from seeing it directly. The only way to get information is to perform a **measurement**, which instantly destroys the superposition and collapses the system to a single classical bitstring. To find the *average* energy, we must:

  1. Prepare the state.
  2. Run the full p-layered circuit.
  3. Measure the outcome.
  4. **Repeat** this entire process thousands of times to build up statistics.

  The average energy is the statistical average over all of these measurement "shots."

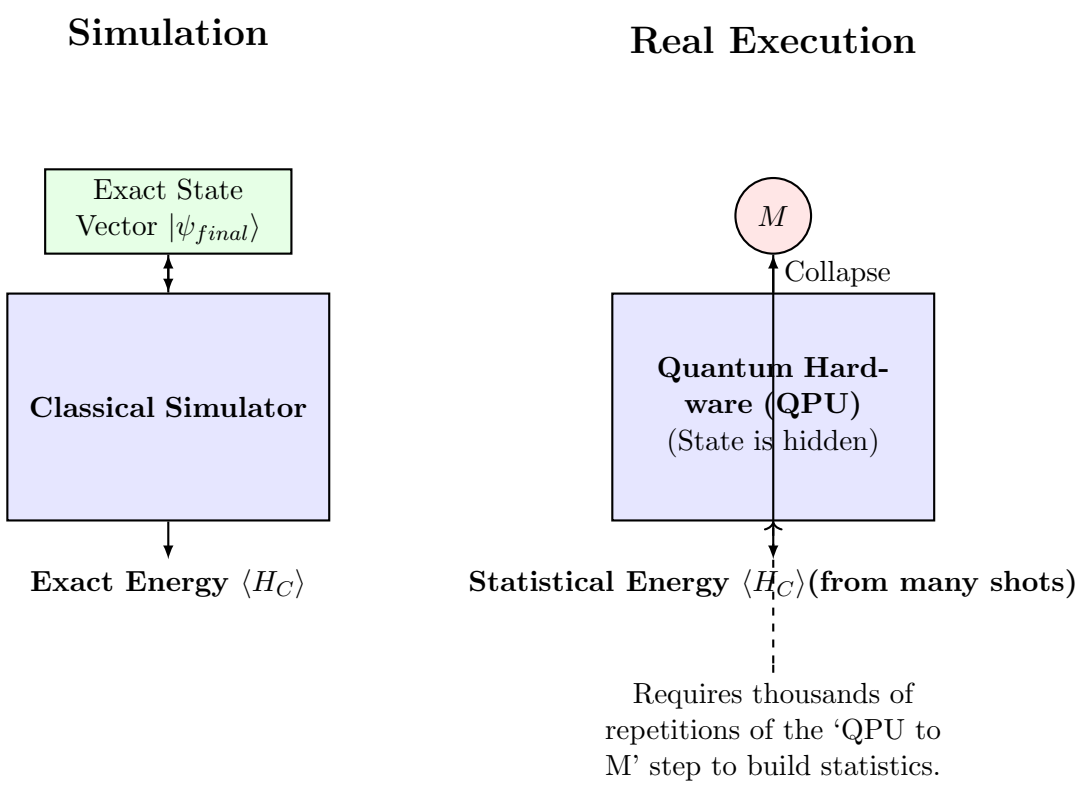


Figure 6: The fundamental difference in information extraction between simulation and real hardware execution.

13 Interpreting QAOA Results

The output of a QAOA run is a list of measurement outcomes. Understanding this output is key.

13.1 Degenerate Ground States and Multiple Solutions

Key Question:

If the ground state is a superposition like $\frac{1}{\sqrt{2}}(|0010\rangle + |0110\rangle)$, what is the solution?

This result indicates that the problem has a **degenerate ground state**, meaning there is more than one optimal solution, and they all have the exact same minimum cost.

- The state $|0010\rangle$ corresponds to one optimal assignment of the variables.

- The state $|0110\rangle$ corresponds to a different but equally optimal assignment.

When you perform the final measurement run (Phase 2), you will measure these two bitstrings with roughly equal high probability (e.g., about 50% each, ignoring noise and algorithmic error). The solution to the problem is therefore the **set of all highly probable outcomes**. This provides richer information than a classical solver that might only return one of the possible solutions.

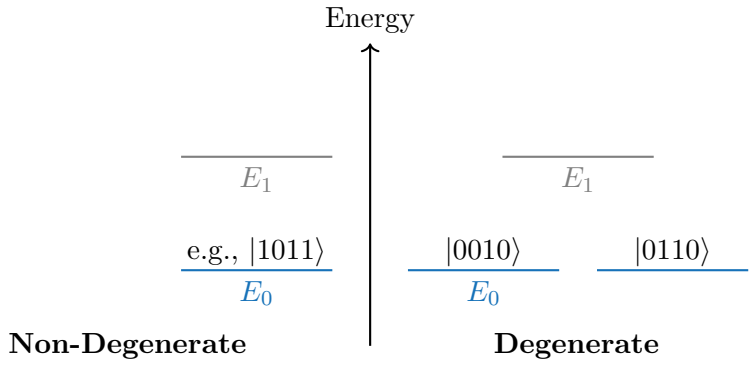


Figure 7: Energy level diagram showing a non-degenerate ground state (left) with a unique solution, versus a two-fold degenerate ground state (right) corresponding to multiple optimal solutions.

13.2 Distinction from Grover's Search

Key Question:

Can we use Grover's search algorithm to find the optimal angles (γ, β)?

No. This is an insightful question that highlights a key distinction. The two algorithms solve fundamentally different kinds of problems.

- **Grover's Search** is for *unstructured search*. It finds a specific, marked item in a discrete list. It requires an "oracle" that can instantly recognize the correct answer. We cannot build such an oracle for the optimal angles because we don't know them in advance.
- **QAOA Parameter Finding** is a problem of *continuous optimization*. We are searching for the best real-numbered values in a continuous landscape that minimize a score (the energy). We don't need to recognize a pre-defined answer; we just need to find the lowest point. This is a task for classical optimizers like COBYLA, not for a search algorithm like Grover's.

Part VI
Inexact Simulation of Quantum Circuits via a DMRG-like Algorithm

14 Motivation: The Challenge of Exact Simulation

The simulation of a quantum circuit on a classical computer is fundamentally limited by the exponential growth of the Hilbert space. A system of N qubits is described by a state vector $|\psi\rangle$ in a Hilbert space of dimension 2^N. An exact simulation requires storing and manipulating this vector, a task whose memory and computational cost scale as $O(2^N)$. This exponential scaling, often called the "tyranny of the Hilbert space," renders the exact simulation of more than $\approx 40 - 50$ qubits intractable, even on the world's largest supercomputers.

Matrix Product States (MPS) provide a powerful alternative by representing the quantum state in a compressed format. An exact simulation using MPS is still possible, but it relies on letting the internal "bond dimension" χ of the MPS grow as needed. For the chaotic and highly-entangling circuits used in quantum algorithms, this bond dimension can grow exponentially with the circuit depth p, i.e., $\chi \sim 2^p$, leading back to an exponential cost.

The goal of an inexact simulation is to circumvent this limitation by accepting a small, controlled error in exchange for keeping the bond dimension χ fixed at a manageable size, $\chi_{\max}$. This approach transforms the simulation cost from being exponential in N or p to being polynomial in N, p, and $\chi_{\max}$, enabling the simulation of much larger systems.

15 The Algorithm: A DMRG-style Compression Step

The algorithm abandons the goal of maintaining the exact state at every step. Instead, it proceeds by evolving the state for a small number of gate layers and then immediately compressing it back down to a maximum allowed bond dimension, $\chi_{\max}$, before proceeding to the next set of layers. This "evolve-then-compress" cycle is inspired by the highly successful Density-Matrix Renormalization Group (DMRG) algorithm from many-body physics.

The core of the method is the **compression step**. Let's assume we have a well-compressed MPS, $|\Psi_{\text{comp}}(D)\rangle$, with bond dimension $\chi_{\max}$, which is our best approximation of the state after D layers of the quantum circuit. The process to advance the state to depth $D+K$ involves the following stages:

1. **Evolve:** We apply a small "batch" of K new gate layers, represented by the unitary operator $U_K = U(D+K)\cdots U(D+1)$, to our current state.

$$|\Psi_{\text{target}}\rangle = U_K|\Psi_{\text{comp}}(D)\rangle \tag{20}$$

Applying the entangling two-qubit gates within U_K creates new correlations between different parts of the quantum state. In the language of MPS, these new correlations must be carried by the virtual bonds connecting the tensors. Consequently, the bond dimension required to exactly represent the state increases. The resulting state, $|\Psi_{\text{target}}\rangle$, is exact relative to its starting point but now has a bond dimension $\chi' > \chi_{\max}$, making it computationally too expensive to store and use in subsequent steps.

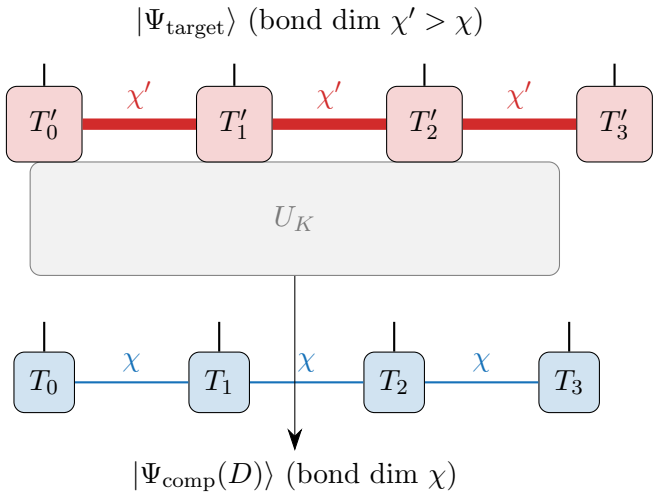


Figure 8: The evolution step. Applying a layer of gates U_K to a compressed MPS increases entanglement, resulting in an uncompressed state with a larger bond dimension χ'.

2. **Compress:** The central task is to find a *new* MPS, $|\Psi_{\text{comp}}(D+K)\rangle$, which adheres to the original small bond dimension $\chi_{\max}$ while being the best possible approximation of the larger, uncompressed state $|\Psi_{\text{target}}\rangle$. This is a variational optimization problem: we seek the state that maximizes the fidelity (the squared overlap).

$$|\Psi_{\text{comp}}(D+K)\rangle = \underset{|\Psi'\rangle \text{ with bond dim } \chi_{\max}}{\operatorname{argmax}} \left|\langle\Psi'|\Psi_{\text{target}}\rangle\right|^2 \tag{21}$$

This optimization problem, while seemingly complex, can be solved efficiently by optimizing one tensor at a time, inspired by the single-site DMRG algorithm. When all tensors in the ansatz $|\Psi'\rangle$ are held fixed except for one, $M'(\tau)$, the problem of maximizing the overlap becomes a simple quadratic optimization. The optimal update for the tensor $M'(\tau)$ is found to be proportional to its "environment tensor," $F(\tau)$. This environment tensor is the contraction of the entire tensor network for the overlap $\langle\Psi'|\Psi_{\text{target}}\rangle$ with the tensor $M'(\tau)$ itself removed. By sweeping back and forth across the chain and iteratively updating each tensor, the algorithm converges to the optimal compressed state.

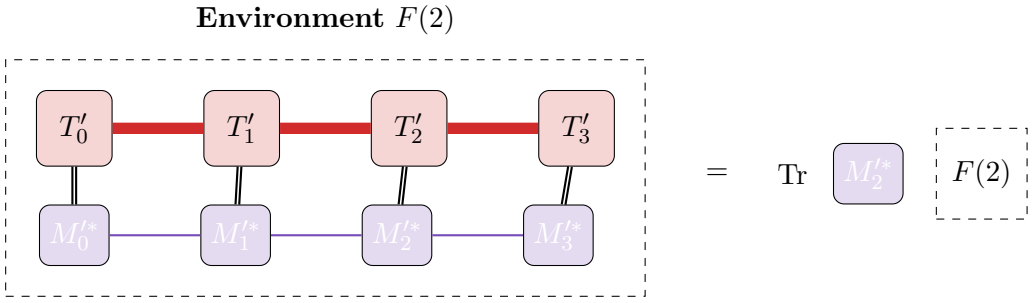


Figure 9: The compression step. To optimize the tensor M'_2, we compute its environment $F(2)$, which consists of the entire overlap network with M'_2 removed. The optimal M'_2 is then found based on $F(2)$.

3. **Iterate:** The newly found compressed state $|\Psi_{\text{comp}}(D+K)\rangle$ is now a high-fidelity, low-complexity representation of the evolved state. It becomes the starting point for the next block of K gate layers. This "evolve-then-compress" cycle is repeated until the entire circuit has been simulated, accumulating a small, quantifiable error at each compression step.

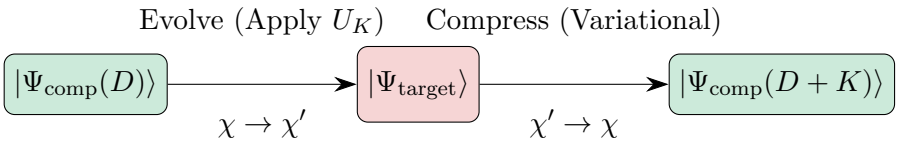


Figure 10: The iterative cycle of the DMRG-style simulation.

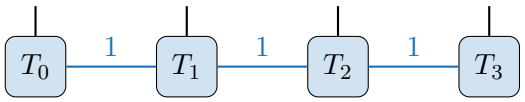
16 Example Comparison: Exact vs. Inexact Simulation

To make the distinction between the exact and inexact simulation methods concrete, let us trace the evolution of a small quantum system under both approaches. We will consider a 4-qubit QAOA circuit with $p = 2$ layers of evolution. For the inexact simulation, we will impose a strict maximum bond dimension of $\chi_{\max} = 2$ and apply compression after each full QAOA layer (i.e., a batch size of $K = 1$). The cost Hamiltonian is the 1D chain $H_C = \sum_{i=0}^{2} Z_i Z_{i+1}$, which is implemented using a "brick-wall" circuit of nearest-neighbor ZZ gates.

16.1 Exact Simulation:

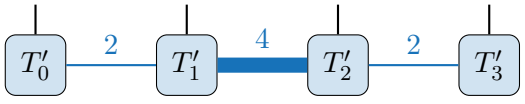
In this mode, accuracy is paramount. The bond dimension of the MPS is allowed to grow as needed to perfectly represent the quantum state at every step.

- **Initial State ($p = 0$):** The simulation begins by applying Hadamard gates to the initial $|0000\rangle$ state, creating an equal superposition $|++++\rangle$. This is a simple product state and is perfectly represented by an MPS with a maximum bond dimension of $\chi = 1$.



Initial State: $\chi_{\max} = 1$

- **After Layer 1 ($p = 1$):** The first layer of entangling gates, $e^{-i\gamma_1 H_C}$, is applied. The "brick-wall" structure first applies ZZ gates to pairs (0,1) and (2,3), and then to pair (1,2). This second step spreads entanglement across the entire chain. To represent this new, more complex state, the bond dimensions must increase. The maximum bond dimension grows to $\chi_{\max} = 4$.



After Layer 1: $\chi_{\max} = 4$

- **After Layer 2 ($p = 2$):** Applying the second entangling layer, $e^{-i\gamma_2 H_C}$, further increases the complexity and entanglement of the state. The bond dimensions must grow again to maintain a perfect representation. The maximum bond dimension roughly doubles again, potentially reaching $\chi_{\max} = 16$.

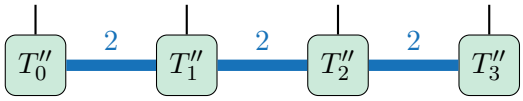
The final state is perfectly accurate, but the bond dimension grows exponentially with the number of layers p, i.e., $\chi \sim 2^{2p}$. This makes the exact simulation method intractable for deep circuits or large systems.

16.2 Inexact Simulation ($\chi_{\max} = 2$, $K = 1$ layer per compression):

In this mode, efficiency is paramount. The bond dimension is strictly forbidden from exceeding $\chi_{\max} = 2$.

- **Initial State ($p = 0$):** The simulation begins identically, with the $|++++\rangle$ state represented by an MPS with $\chi_{\max} = 1$.
- **Compression Step 1 (after Layer 1):**
  1. *Evolve:* Apply the first full QAOA layer (both U_C and U_B) to the initial state. As determined by the exact simulation, the resulting uncompressed state, $|\Psi_{\text{target}}(1)\rangle$, requires a maximum bond dimension of $\chi = 4$ for an exact representation.
  2. *Compress:* This uncompressed state is now the target for our variational algorithm. We seek the best possible approximation, $|\Psi_{\text{comp}}(1)\rangle$, that can be represented with our constraint of $\chi_{\max} = 2$. This is achieved by first performing a fast SVD truncation to get an initial guess, and then refining this guess with the iterative DMRG-style sweeps. This step introduces a small, controlled error. The partial fidelity of this step, $f_1 = |\langle\Psi_{\text{comp}}(1)|\Psi_{\text{target}}(1)\rangle|^2$, will be slightly less than 1.

  After this step, our working MPS, now $|\Psi_{\text{comp}}(1)\rangle$, has its bond dimension capped at 2.



After Compression 1: $\chi_{\max} = 2$

- **Compression Step 2 (after Layer 2):**
  1. *Evolve:* Apply the second QAOA layer to our current compressed state, $|\Psi_{\text{comp}}(1)\rangle$. The gate applications will again cause the bond dimension to grow beyond our limit, resulting in a new uncompressed state $|\Psi_{\text{target}}(2)\rangle$ with $\chi \approx 4$.
  2. *Compress:* Run the variational compression algorithm again to find the best approximation $|\Psi_{\text{comp}}(2)\rangle$ with $\chi_{\max} = 2$. This introduces another small error, and we calculate a new partial fidelity, $f_2 < 1$.

The final state, $|\Psi_{\text{comp}}(2)\rangle$, is an approximation, but its bond dimension was kept constant throughout the simulation. The cost has been shifted from memory (storing exponentially large tensors) to computation (performing the repeated variational compressions). The final estimated fidelity is the product of the partial fidelities from each step, $F_{\text{total}} \approx f_1 \times f_2$.

17 Fidelity Estimation without the Exact State

For any large-scale inexact simulation, the true, exact state vector $|\Psi_{\text{exact}}\rangle$ is computationally unreachable. A practical method must therefore be able to estimate the accuracy of its results without reference to this unknowable state. The DMRG-style simulation achieves this through the **multiplicative fidelity law**, which provides a remarkably accurate, internally consistent estimate of the final state's fidelity.

17.1 The Multiplicative Fidelity Law:

The core principle is that the total fidelity of the simulation after m compression steps, F_{total}, can be approximated by the product of the "partial fidelities," f_δ, from each individual compression step δ.

$$F_{\text{total}} = |\langle \Psi_{\text{exact}}(m) | \Psi_{\text{comp}}(m) \rangle|^2 \approx \tilde{F} = \prod_{\delta=1}^{m} f_\delta \tag{22}$$

Here, $\tilde{F}$ is our computable estimate of the total fidelity. The partial fidelity f_δ is defined as the squared overlap between the uncompressed state after a batch of gates and the new, compressed state, a quantity that is calculated during the simulation itself:

$$f_\delta = |\langle \Psi_{\text{comp}}(\delta) | \Psi_{\text{target}}(\delta) \rangle|^2 \tag{23}$$

17.2 Detailed Mathematical Derivation:

To understand why this approximation is so effective, we must analyze how the error from one compression step propagates to the next. Let us define the key states at a given step δ:

- $|\Psi_{\text{exact}}(\delta)\rangle$: The true, exact state after δ layers of gates.
- $|\Psi_{\text{comp}}(\delta)\rangle$: The compressed MPS representation of the state after step δ.
- $|\Psi_{\text{target}}(\delta)\rangle$: The intermediate, uncompressed state obtained by applying the δ-th batch of gates, U_δ, to the previous compressed state: $|\Psi_{\text{target}}(\delta)\rangle = U_\delta |\Psi_{\text{comp}}(\delta - 1)\rangle$.

The error at the previous step, $\delta - 1$, is the difference vector between the exact and compressed states:

$$|\varepsilon_{\delta-1}\rangle = |\Psi_{\text{exact}}(\delta - 1)\rangle - |\Psi_{\text{comp}}(\delta - 1)\rangle \tag{24}$$

The exact state at step δ can be written by evolving the state from the previous step:

$$\begin{aligned} |\Psi_{\text{exact}}(\delta)\rangle &= U_\delta |\Psi_{\text{exact}}(\delta - 1)\rangle && (25)\\ &= U_\delta \left(|\Psi_{\text{comp}}(\delta - 1)\rangle + |\varepsilon_{\delta-1}\rangle \right) && (26)\\ &= U_\delta |\Psi_{\text{comp}}(\delta - 1)\rangle + U_\delta |\varepsilon_{\delta-1}\rangle && (27)\\ &= |\Psi_{\text{target}}(\delta)\rangle + U_\delta |\varepsilon_{\delta-1}\rangle && (28) \end{aligned}$$

Now, let's examine the total overlap at step δ:

$$\begin{aligned} \langle \Psi_{\text{exact}}(\delta) | \Psi_{\text{comp}}(\delta) \rangle &= \left\langle \left(|\Psi_{\text{target}}(\delta)\rangle + U_\delta |\varepsilon_{\delta-1}\rangle \right) \right| |\Psi_{\text{comp}}(\delta)\rangle && (29)\\ &= \underbrace{\langle \Psi_{\text{target}}(\delta) | \Psi_{\text{comp}}(\delta) \rangle}_{\text{Partial Overlap}} + \underbrace{\langle U_\delta \varepsilon_{\delta-1} | \Psi_{\text{comp}}(\delta) \rangle}_{\text{Propagated Error Term}} && (30) \end{aligned}$$

The multiplicative law, $F_{\text{total}}(\delta) \approx F_{\text{total}}(\delta - 1) \cdot f_\delta$, holds if and only if the second term—the overlap of the evolved previous error with the current compressed state—is negligible.

17.3 The Physical Justification for the Approximation:

The validity of neglecting the propagated error term rests on a powerful physical argument concerning the nature of chaotic quantum evolution and the structure of Matrix Product States.

1. **The Structure of the Compressed State:** The state $|\Psi_{\text{comp}}(\delta)\rangle$ is an MPS constrained by a small bond dimension χ. Such states are known to be low-entanglement states that occupy a vanishingly small, highly structured corner of the total Hilbert space. This is often referred to as the "low-entanglement manifold."

2. **The Structure of the Error Vector:** The error vector $|\varepsilon_{\delta-1}\rangle$ is not random. It is the projection of the exact state onto the subspace that was discarded during the $(\delta - 1)$-th compression. It is composed of the Schmidt states corresponding to the smallest, discarded singular values. As such, it is also a highly structured, low-entanglement vector.

3. **The Effect of Chaotic Evolution:** For the pseudo-random and highly entangling circuits typical of quantum algorithms (like QAOA or supremacy benchmarks), the unitary evolution U_δ acts as a powerful "scrambler" or quantum pseudo-randomizer. When applied to the structured, low-entanglement error vector $|\varepsilon_{\delta-1}\rangle$, it rapidly destroys its structure and "smears" its information content across an exponentially large number of basis states. The resulting vector, $U_\delta|\varepsilon_{\delta-1}\rangle$, is effectively a generic, high-entanglement state that appears as random noise, similar to a vector drawn from the Porter-Thomas distribution.

Therefore, the propagated error term represents the overlap between a generic, chaotic, high-entanglement vector ($U_\delta|\varepsilon_{\delta-1}\rangle$) and a specific, structured, low-entanglement vector ($|\Psi_{\text{comp}}(\delta)\rangle$). The probability of two such vectors having a significant overlap is exponentially small in the system size. The error from one step is effectively "randomized" into a part of the Hilbert space that is orthogonal to the space of compressible states where the next step's solution resides.

This effective orthogonalization of the propagated error makes the errors at each step nearly independent. This independence is the key justification for why the total fidelity can be so accurately approximated by the product of the partial fidelities. The partial fidelity f_δ is a quantity that can be calculated directly at each step of the simulation, as it is the final value of the cost function being optimized by the variational compression algorithm itself. This provides a practical and accurate method for estimating the fidelity of a simulation even when the exact answer is computationally unreachable.‴

Part VII
Summary and Conclusion

18 Overview of the Journey

This document has detailed a comprehensive journey from the challenges of classical quantum simulation to the principles and practicalities of modern quantum optimization algorithms. The discussion progressed through three major themes:

1. **Efficient Classical Simulation for 1D Systems:** The limitations of full state vector simulation for many-body systems led to the development of tensor network methods. We explored the Time-Evolving Block Decimation (TEBD) algorithm, which leverages Matrix Product States (MPS) to efficiently simulate the time evolution of 1D systems with limited entanglement. The core mechanism involves a Trotter-Suzuki decomposition of the evolution operator and a repeated, controlled truncation of the state using the Singular Value Decomposition (SVD).

2. **The Quantum Approximate Optimization Algorithm (QAOA):** As a primary method for solving combinatorial optimization problems on near-term quantum computers, we deconstructed QAOA from its theoretical origins in Adiabatic Quantum Computation (AQC). We detailed its structure as a hybrid quantum-classical variational algorithm, comprising:

   - A **Cost Hamiltonian (H_C)** that encodes the classical problem into its energy spectrum.
   - A **Mixer Hamiltonian (H_B)** that drives exploration of the solution space.
   - A layered quantum circuit (the "ansatz") with $2p$ trainable angles $(\boldsymbol{\gamma}, \boldsymbol{\beta})$.
   - A **classical optimization loop** that iteratively refines these angles to minimize the measured energy of the prepared state.

3. **Practical Implementation and Advanced Concepts:** To bridge theory and practice, we covered the full workflow of applying QAOA. This included the crucial mapping of classical problems into the **QUBO** format and subsequently into an **Ising Hamiltonian**. We discussed the necessity of **Sparse Pauli Operators** for representing these Hamiltonians in software. To address the practical scaling challenges of standard QAOA, the **Recursive QAOA (R-QAOA)** was introduced as a powerful "divide and conquer" heuristic. Finally, we clarified the fundamental operational differences between classical simulation (direct state vector access) and execution on real quantum hardware (reliance on statistical "shots" due to measurement collapse).

4. **Advanced Simulation of Quantum Algorithms:** Finally, to bridge the gap between classical simulation capabilities and the demands of quantum algorithms like QAOA, we developed a powerful **inexact simulation** technique inspired by the Density-Matrix Renormalization Group (DMRG). Where TEBD uses a simple, greedy SVD truncation, this advanced method trades perfect fidelity for immense computational scalability. Its core components are:

   - An "evolve-then-compress" cycle, where the state is evolved under a small batch of gates, temporarily increasing its bond dimension.
   - An immediate compression of the state back to a fixed, manageable bond dimension using an iterative **variational compression** algorithm, which finds the globally optimal MPS approximation at each step.
   - A method to estimate the final simulation fidelity via a **multiplicative law**, which leverages the fact that chaotic quantum evolution effectively randomizes the errors introduced at each compression step, making it possible to track accuracy without access to the true, exponentially large state vector.

19 Key Principles Revisited

Throughout our discussion, several key principles emerged as central to understanding these algorithms:

- **Entanglement as a Resource and a Bottleneck:** In TEBD, the ability of MPS to capture area-law entanglement is the key to its success. Its limitation arises when entanglement becomes too large (violating the area law), at which point the required bond dimension grows exponentially. Similarly, the power of quantum algorithms like QAOA lies in their ability to create and manipulate highly entangled states that are intractable to simulate classically.

- **Approximation is Key:** None of these algorithms are exact solvers for all problems. TEBD relies on Trotter and truncation approximations. QAOA for a finite p is an *approximate* optimization algorithm. R-QAOA is a *heuristic* approximation. The goal is to find a "good enough" solution for classically intractable problems within a reasonable amount of time and resources.

- **The Hybrid Quantum-Classical Paradigm:** Near-term quantum algorithms are not purely quantum. They are powerful because they delegate tasks to the most suitable processor. The quantum computer performs the task it excels at: preparing and evolving complex superpositions. The classical computer performs the tasks it is best at: control, data processing, and complex optimization loops. The synergy between the two is what makes algorithms like QAOA and VQE practical.

- **Measurement is the Interface:** The act of measurement is the only bridge between the hidden quantum state and the classical world. Understanding its probabilistic nature and the necessity of repeated shots to build statistical estimates of observables like energy is fundamental to understanding how all near-term quantum algorithms operate on real hardware.

- **Simulation as a Benchmark for Quantum Hardware:** Our development of an advanced, inexact classical simulation method underscores a modern principle: the benchmark for "quantum advantage" is a moving target. The DMRG-style variational simulation acts as a powerful classical analogue to a noisy quantum computer. Both operate with a finite, quantifiable fidelity, and both face a trade-off between the complexity of the task and the accuracy of the result. By pushing the limits of what is classically simulable with controlled approximations, we provide a concrete and challenging benchmark that near-term quantum hardware must surpass to demonstrate true, practical advantage.

This concludes our detailed notes on this topic. The path from TEBD to R-QAOA demonstrates a clear arc in computational physics and computer science: from developing efficient classical methods that work within certain constraints, to designing novel quantum algorithms inspired by those same physical principles, and finally, to refining those quantum algorithms with clever heuristics to make them practical on the hardware of today and tomorrow.