Quantum Optimization and the Traveling Salesman Problem

Matthew Torre and Kai Roybal

Table of contents

01

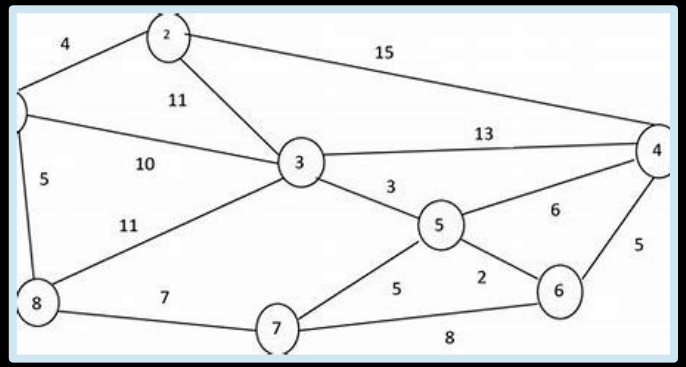
The Traveling Salesman Problem

A Brief Introduction

The Traveling Salesman Problem

Given a list of cities and distances between pairs of cities, the Traveling Salesman Problem looks to find the shortest possible path to visit each city at least once and return to the original city.

In combinatorial optimization and theoretical computer science, the Traveling Salesman Problem (TSP) was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods.



The Traveling Salesman Problem Cont.

This problem represents a combinatorial optimization problem and is especially important as it belongs to NP-complete problems. If a truly efficient solution is found, then all problems a part of the NP-complete class would be solved.

A well known puzzle due to the simplicity in explaining it and the complexity in actually solving it. While traveling salesman are obscure today, many problems can be generalized as TSPs or restructured as one.

TSP Type Problems

- Analysis of crystal structures
- Overhauling of gas turbine engines
- In material handling in a warehouse
- Cutting stock problems
- Clustering of data arrays
- Assigning plane fleet routes
- Genome sequencing

Computational Complexity

The question we ask here is:

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

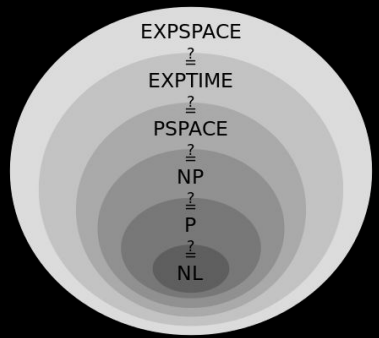
NP-Hard Problems

Definition: The class NP (Nondeterministic Polynomial time) consists of problems for which a given solution can be verified in polynomial time.

Definition: A problem is NP-Hard if it is at least as hard as the hardest problems in NP.

- No known polynomial-time algorithm to solve all instances for TSP.

Note: It is possible that the worst-case running time for any algorithm for the TSP increases superpolynomial (but no more than exponentially) with the number of cities.



TSP’s Computational Complexity

NP-hard:

- No known polynomial-time algorithm to solve all instances.

Exact Algorithms:

- Require exponential time.
- Example: Dynamic Programming (Bellman-Held-Karp): $O(n\hat{}2 \times 2n)$

Approximation Algorithms:

- Provide near-optimal solutions.
- Example: Christofides' Algorithm: Guarantees within 1.5 times the optimal solution

02

Quantum States

Properties of Qubits, Quantum Gates, Superposition, Entanglement

Quantum States

Qubits:
Basic unit of quantum information.
Can be in a superposition of 0 and 1.

- Represent the problem's solution space.

Superposition:
A qubit can represent both 0 and 1 simultaneously.

- Allows simultaneous exploration of multiple solutions.

Entanglement:
Qubits can be entangled, meaning the state of one qubit can depend on the state of another.

- Captures dependencies between different parts of the problem.

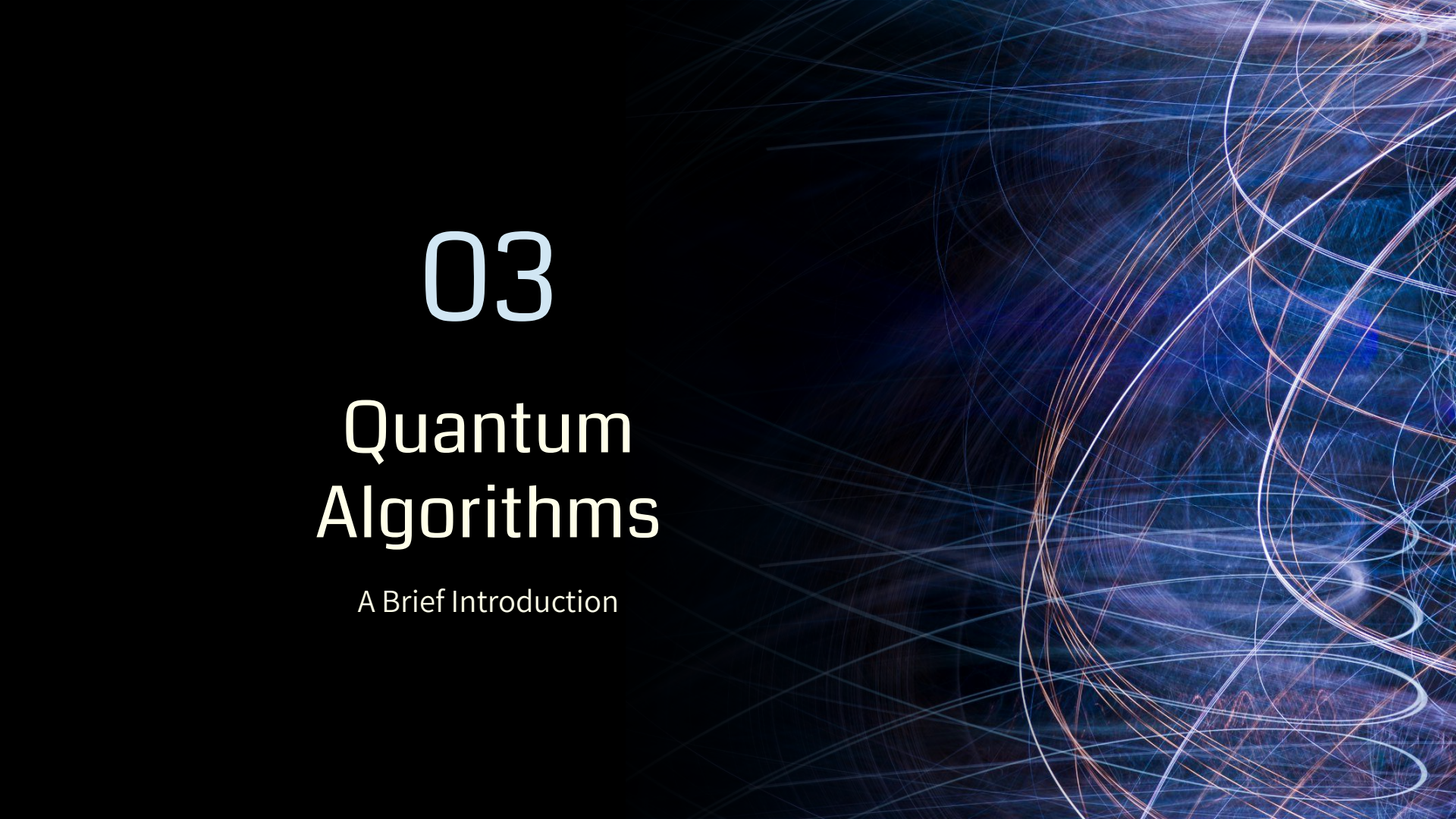
Quantum Gates:
Operations that change the state of qubits.
Examples: Pauli-X, Hadamard, CNOT.

- Implement the operations necessary for evolving the quantum state in QAOA to find optimal or near-optimal solutions for TSP.

03

Quantum Algorithms

A Brief Introduction

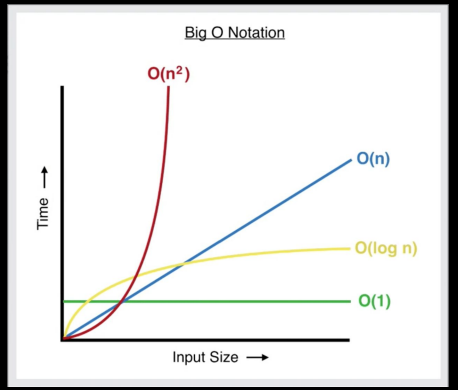


Explaining Quantum Algorithms

Goal of quantum algorithms is to be faster than classical counterparts while utilizing quantum superposition and entanglement.

- Use less gates
- Efficient: polynomial (linear) circuit complexity
  - Least number of gates to implement
- Easier to determine query complexity (Big O)
  - Number of calls to function

- These algorithms look like the quantum circuits we have already seen.
- Quantum Tortoise vs Classical Hare
  - Which path from A to B is better?



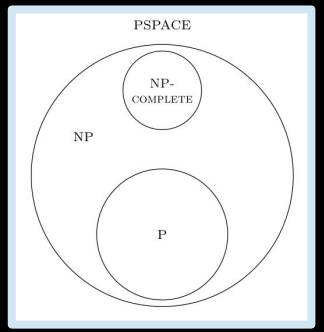
Explaining Quantum Algorithms

Quantum algorithms are used to solve different problems of different complexity classes:

P Class: problems solved in polynomial (linear) time by classical computers
- Determining if a number is prime

NP Class: problems in which a computer can quickly verify the solution in polynomial time
- Completeness
- Factoring

P Space: problems that can be solved with polynomial amount of memory regardless of time
- Finding winning strategy for checkers
- Winning a general level of Mario

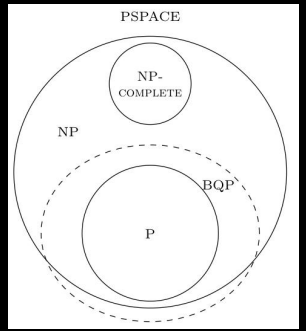


Explaining Quantum Algorithms

Turing Machine - Allows computer scientists to study the power and limitations of a computer

Strong Church-Turing Thesis: An efficient algorithm on any computer is efficient on a probabilistic Turing Machine

Quantum computers hope to violate this thesis in order to solve problems efficiently that are otherwise inefficient to do classically. While there is no proof this is possible, there is strong evidence, with one example being Shor's algorithm. In all the examples that provide evidence, quantum computers are superpolynomially faster than classical algorithms.



Quantum Algorithms Overview

Famous Quantum Algorithms

Shor's Algorithm

Shor's Algorithm efficiently factors large integers into their prime components, solving a problem believed to be hard for classical computers and forming the basis of RSA encryption. By using the Quantum Fourier Transform and period finding, it can factor numbers in polynomial time, showcasing a profound quantum speedup over classical methods.

Grover's Algorithm

Grover's Algorithm provides a quadratic speedup for searching an unsorted database by using amplitude amplification to increase the probability of finding the desired element. This algorithm can solve search and optimization problems in O(sqrt(N) time, significantly faster than the O(N) time required classically.

QAOA

The Quantum Approximate Optimization Algorithm (QAOA) is designed to solve combinatorial optimization problems by alternating quantum and classical computations. It encodes the problem into a quantum state using cost and mixer Hamiltonians and iteratively optimizes parameters to find an approximate solution efficiently.

Shor's Algorithm

Problem: number N that is product of prime numbers p and q. Find factors p and q.

Classical solution: *number field sieve*.

- Slower than polynomial run time and is therefore inefficient.

Quantum Solution with Shor's Algorithm:

- Best evidence that quantum algorithms can be better than classical
- RSA cryptography

$$p = \gcd(a^{r/2} - 1, N),$$
$$q = \gcd(a^{r/2} + 1, N).$$

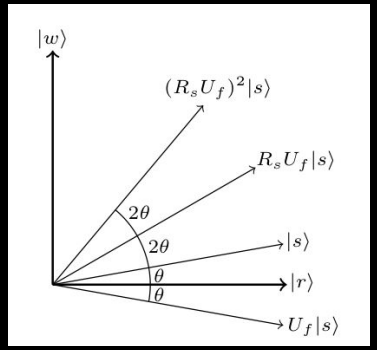
Grover's Algorithm

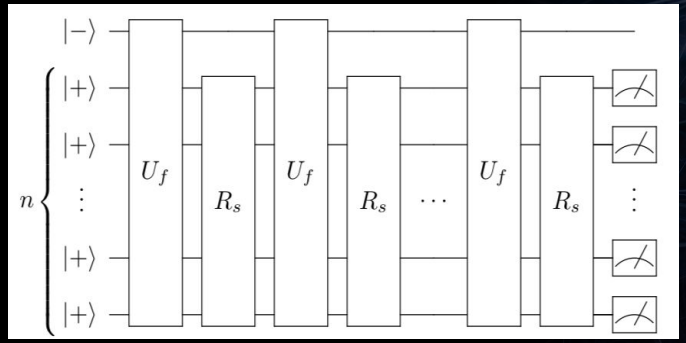
Problem: Brute force searching. Ex: phone book (inverse)

Classical solution: O(N)

Quantum Solution with Grover's Algorithm:

- Quadratic speedup

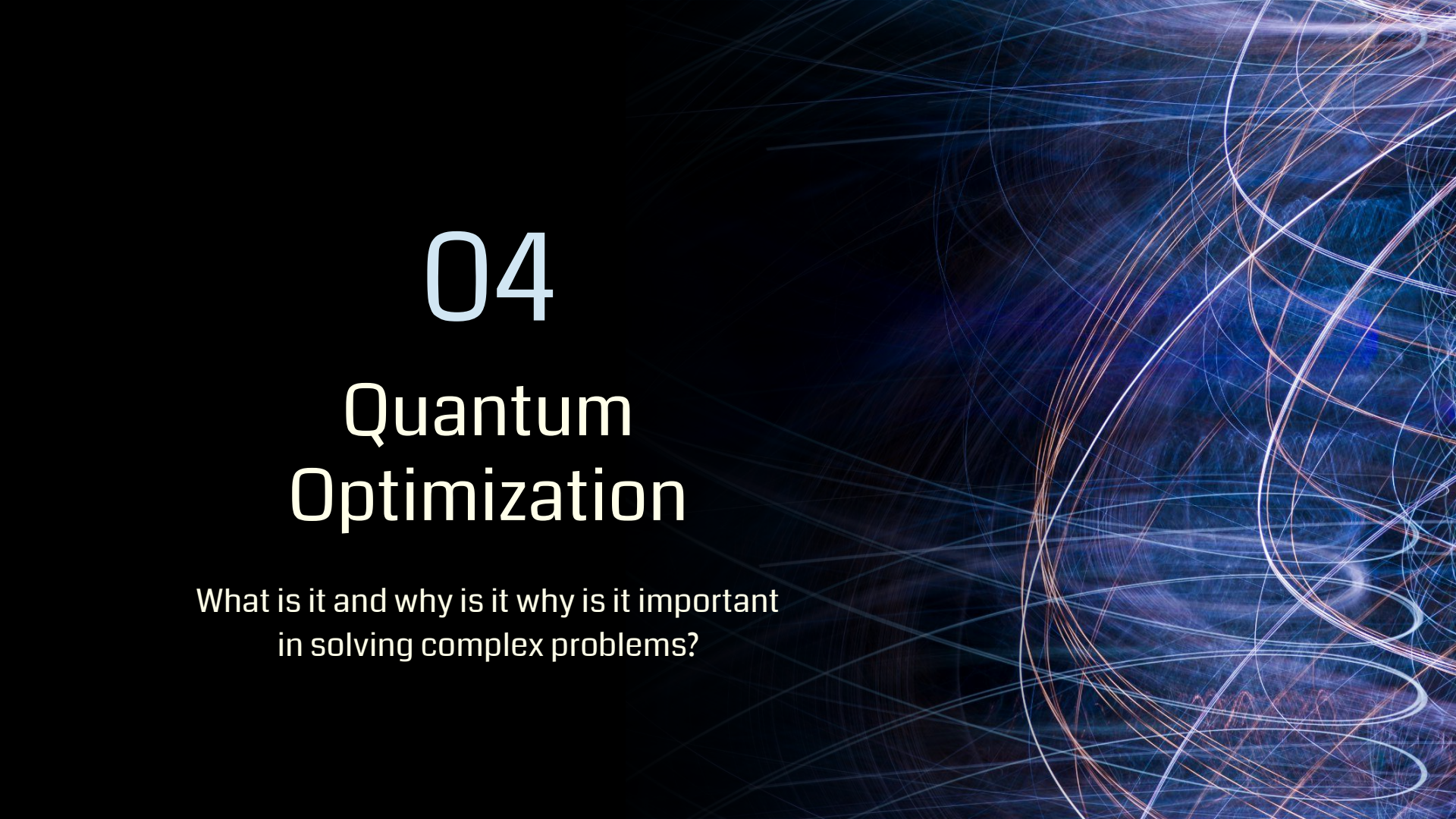




04

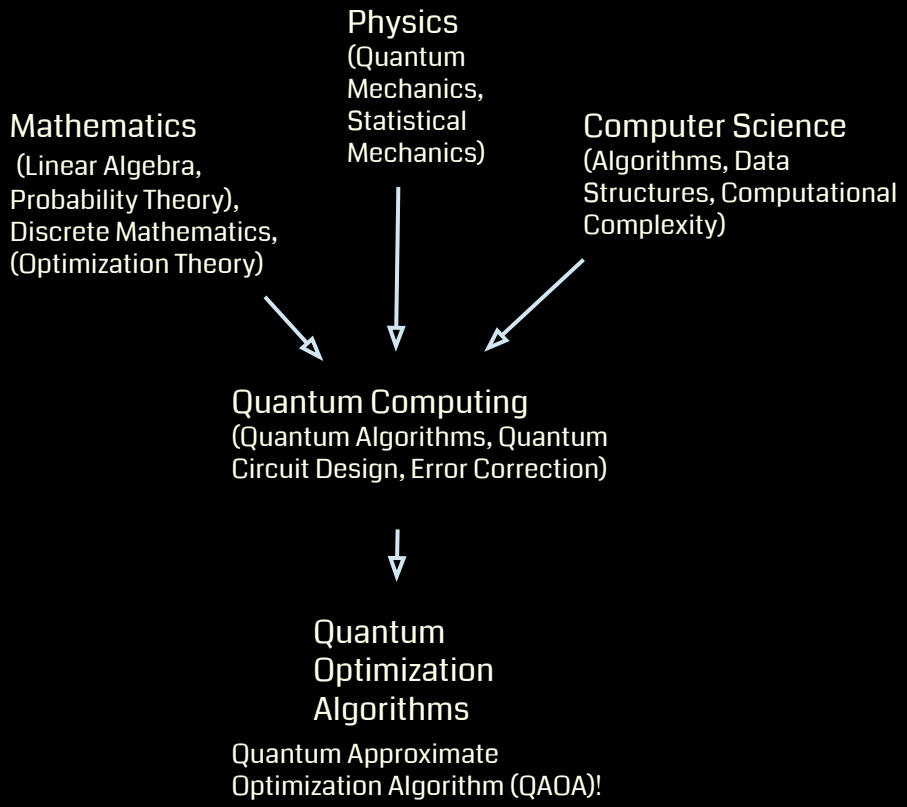
Quantum Optimization

What is it and why is it why is it important in solving complex problems?



Quantum Optimization Algorithms

A Bird's Eye View



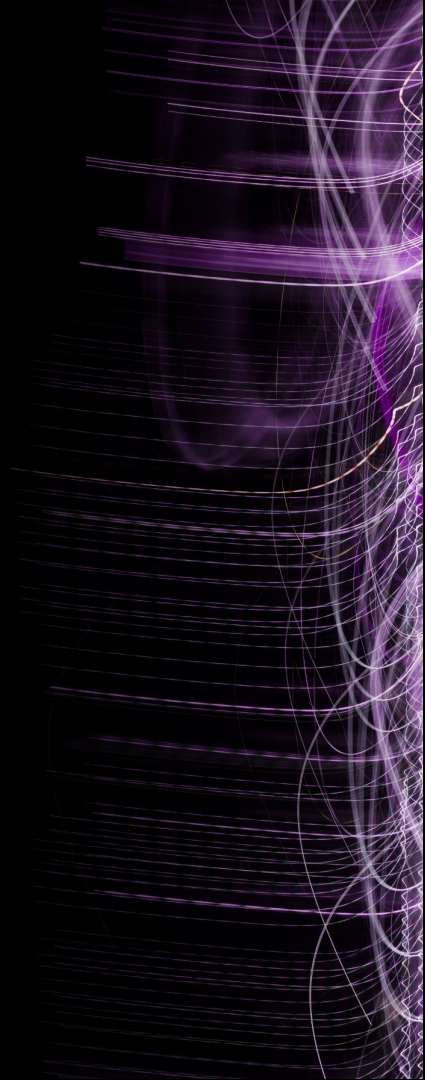
Introduction to QAOA

Purpose: QAOA is used to find approximate solutions to combinatorial optimization problems.

Key Idea: Combines quantum and classical techniques to optimize a cost function.

Note:
Combinatorial Optimization Problems: These are problems where the objective is to find the best solution from a finite set of possible solutions. We'll be solving the Traveling Salesman Problem (TSP) but other examples include the Max-Cut problem, and Minimum spanning tree.

Quantum and Classical Techniques: QAOA leverages the strengths of quantum computation (entanglement) and classical optimization methods (iterative improvement) to find good solutions.



QAOA Prelude

The Quantum Approximate Optimization Algorithm (QAOA) is designed to solve combinatorial optimization problems.

By combining quantum and classical techniques to find approximate solutions to problems.

Let's break it down from first principles...

Stepping through the Algorithm

Problem Setup:

Objective: Optimize an objective function that represents the cost.

- **Objective**: Optimize an objective function $C(x)$ for $x \in \{0,1\}^n$.
- **Hamiltonians**:
  - Cost Hamiltonian H_C encoding $C(x)$.
  - Mixer Hamiltonian $H_M = \sum_{i=1}^{n} X_i$, where X_i are Pauli-X operators.

Problem Setup Cont.

Objective Function $C(x)$: A function that assigns a cost to each possible solution x. The goal is to find the solution that minimizes (or maximizes) this cost.

Binary String x: In QAOA, the solutions are represented as binary strings of length n. Each bit in the string can be either 0 or 1.

Hamiltonians: In quantum mechanics, a Hamiltonian is an operator corresponding to the total energy of the system.

Cost Hamiltonian H_C : Encodes the objective function into a form that a quantum computer can process.

Mixer Hamiltonian H_M: Helps explore different possible solutions by mixing the states. The Pauli-X operator: Xi flips the state of the i-th qubit and equates to the to a rotation around the x axis on the Bloch Sphere

- Purpose: The mixer Hamiltonian generates a superposition of states, enabling the quantum system to explore multiple potential solutions simultaneously.

Initialization

Quantum State $|\psi\_0\rangle$: A quantum state is a vector that describes the state of a quantum system. Here, $|\psi\_0\rangle$ is the initial quantum state.

Explanation: This state is a superposition of all possible binary strings of length n

Prepare the Initial State:

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

Constant Term: Normalization factor to ensure the probability of all states sums to 1. $\frac{1}{\sqrt{2^n}}$

Cost and Mixer Unitaries

Unitary Operators: In quantum mechanics, unitary operators are used to describe the evolution of quantum states. They preserve the norm of the quantum state.

Cost Unitary: $U(C,\gamma)$Applies the cost Hamiltonian H_C to the quantum state for a time duration proportional to γ (gamma). This operation encodes the problem into the quantum state.

Cost Unitary:

$$U(C,\gamma) = e^{-i\gamma H_C}$$

Mixer Unitary:

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

Explanation: These unitaries evolve the quantum state according to the cost and mixer Hamiltonians.

Cost and Mixer Unitaries cont.

Exponential of a Matrix: The operation $e^{\wedge}(-i\gamma H\_C)$ is the matrix exponential, which describes how the quantum state evolves over time under the influence of the Hamiltonian H_C.

$$U(C,\gamma) = e^{-i\gamma H_C}$$

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

Mixer Unitary U(B,β): Applies the mixer Hamiltonian H_M to the quantum state for a time duration proportional to β.

Purpose: This operation helps explore different possible solutions by mixing the states.

Forming the Quantum State

Explanation: Alternating applications of cost and mixer unitaries form the quantum state.

This expression represents the final quantum state after applying p layers of alternating unitaries

State After p Layers:

$$|\psi(\vec{\gamma},\vec{\beta})\rangle = U(B,\beta_p)U(C,\gamma_p)\cdots U(B,\beta_1)U(C,\gamma_1)|\psi_0\rangle$$

These represent a vector of parameters for the cost and mixer unitaries respectively. These are crucial for as they determine how long each unitary operation is applied

Apply the First Cost Unitary U(C,γ _1): This encodes the cost function into the quantum state.

Apply the First Mixer Unitary U(B,β_1): This explores the solution space by mixing the states.

Repeat the Process: Alternate applications of cost and mixer unitaries for p players.

This state is typically an equal superposition of all possible states.

Measurement and Optimization

Measure the Final State: Measure the Final State: $|\psi(\gamma,\beta)\rangle$

Expectation Value: ⟨C⟩=⟨ψ(γ,β)|H_C|ψ(γ,β)⟩

Classical Optimization: Adjust parameters γ and β to maximize $\langle C\rangle$.

Measure the Final State: $|\psi(\vec{\gamma},\vec{\beta})\rangle$

Expectation Value:

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

Classical Optimization: Adjust parameters

Represents the average value of the cost function when measured over the quantum state.

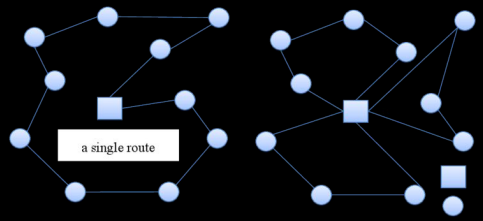
Involves iteratively adjusting the parameters γ and β to optimize the expectation value ⟨C⟩

Measurement: Involves collapsing the quantum state |ψ(γ, β)⟩ to obtain a classical outcome.

Quantum Approximate Optimization Algorithm (QAOA)

Algorithm

1) Initialize: Prepare the superposition state.
2) Alternating Unitaries: Apply p layers of cost and mixer unitaries.
3) Measure: Measure the final state.
4) Optimize: Optimize parameters using classical algorithms.
5) Iterate: Repeat until optimal parameters are found.
6) Output: Measure the optimized state to obtain the solution.



Quantum Approximate Optimization Algorithm (QAOA)

1. Problem Setup

- **Objective**: Optimize an objective function $C(x)$ for $x \in \{0,1\}^n$.
- **Hamiltonians**:
  - Cost Hamiltonian H_C encoding $C(x)$.
  - Mixer Hamiltonian $H_M = \sum_{i=1}^{n} X_i$, where X_i are Pauli-X operators.

2. Initialize

- Prepare the initial state:

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

3. Alternating Unitaries

- Apply the cost unitary p times with parameters $\vec{\gamma}$:

$$U(C, \gamma) = e^{-i\gamma H_C}$$

- Apply the mixer unitary p times with parameters $\vec{\beta}$:

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

- Form the state after p layers:

$$|\psi(\vec{\gamma}, \vec{\beta})\rangle = U(B, \beta_p)U(C, \gamma_p) \cdots U(B, \beta_1)U(C, \gamma_1)|\psi_0\rangle$$

4. Measurement and Optimization

- Measure the final state $|\psi(\vec{\gamma}, \vec{\beta})\rangle$ to obtain a candidate solution.
- Calculate the expectation value of the cost function:

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

- Use a classical optimization algorithm to adjust the parameters $\vec{\gamma}$ and $\vec{\beta}$ to maximize $\langle C \rangle$.

5. Iterate

- Repeat steps 2-7 until convergence on the optimal parameters $\vec{\gamma}^*$ and $\vec{\beta}^*$.

6. Output

- The measurement of the state $|\psi(\vec{\gamma}^*, \vec{\beta}^*)\rangle$ yields the approximate solution to the optimization problem.

Why Optimization?

Quantum computing has the potential to revolutionize many aspects of society, with optimization being a key application area. By leveraging quantum algorithms, we can streamline complex processes, to achieve significant improvements over classical solutions. Optimization problems are ubiquitous in industries such as logistics, finance, and engineering.

05

The Traveling Salesman Problem Revisited

Demonstration

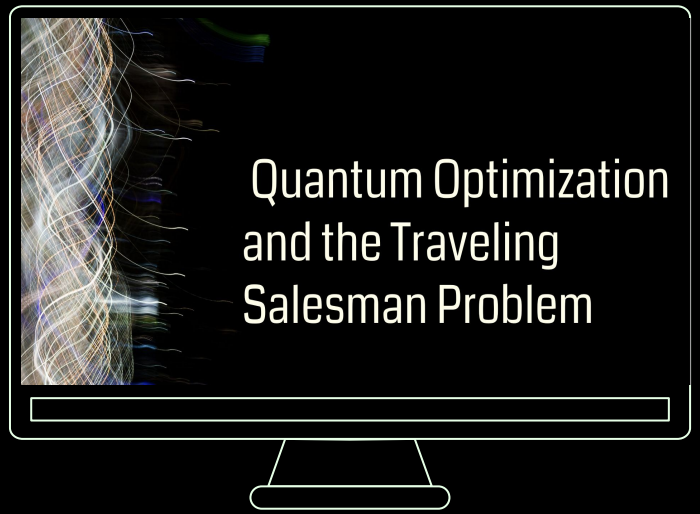
Demonstration

Let’s Watch QAOA in action!

Demonstration

We will test 3 different specifications of the problem:

1) 4 Cities & Their Respective Distances
2) 8 Cities & Their Respective Distances
3) 15 Cities & Their Respective Distances

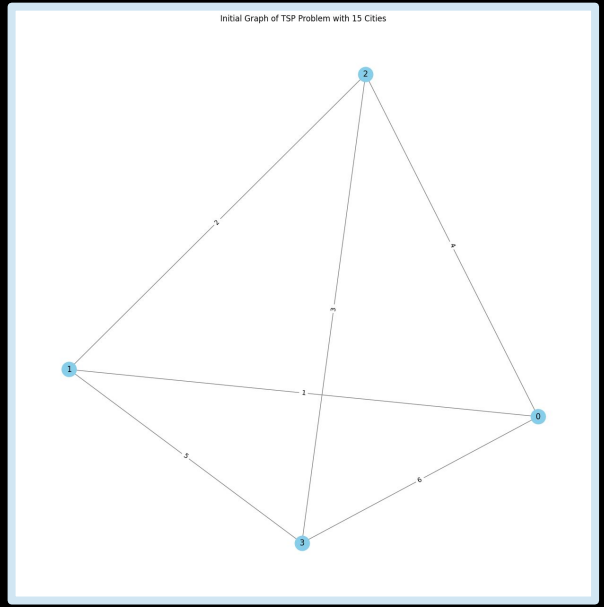


Cirq is a Python library for building, transforming, and simulating quantum circuits on real or simulated devices

Problem Formulation: Phase 1

Cities and Distances:
Cities: [0, 1, 2, 3]

Distances: {(0, 1): 1, (0, 2): 4, (0, 3): 6, (1, 2): 2, (1, 3): 5, (2, 3): 3}

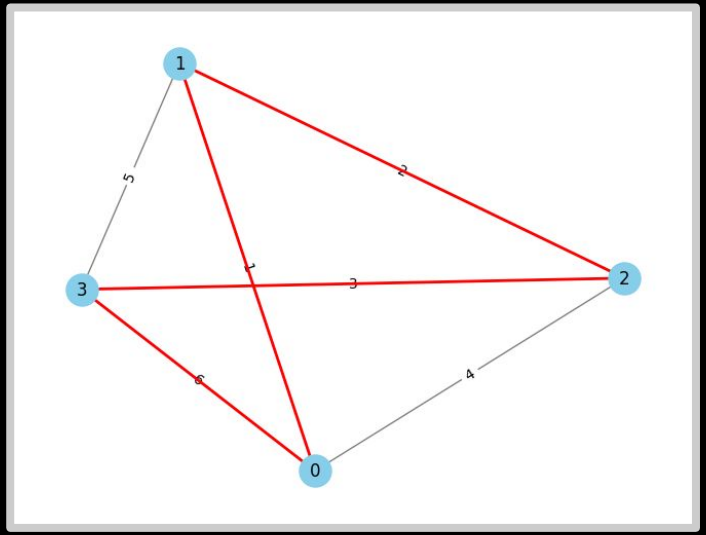


Results

How can we interpret our results?

The most common bitstring line gives us the binary string that most frequently represents the result of running the quantum algorithm.

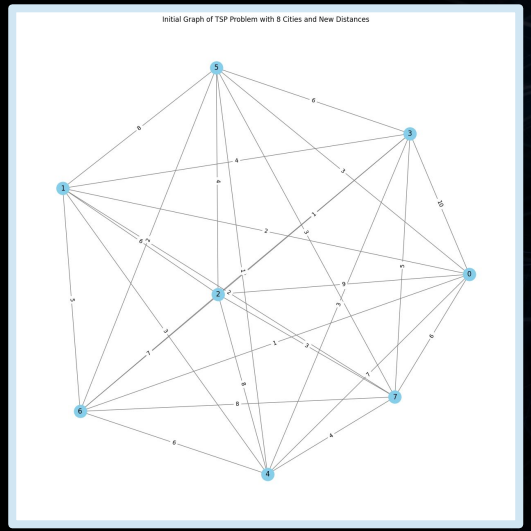
The bitstring is then mapped to a tour sequence based on our implemented logic.



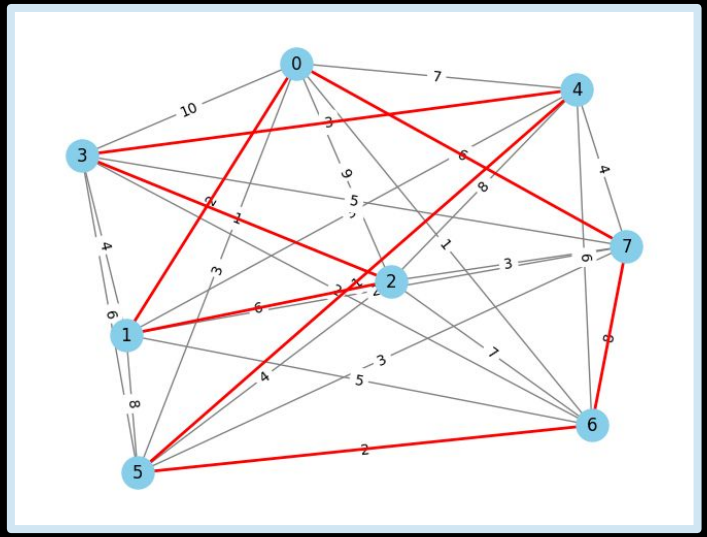
Most common bitstring: 1111
TSP Tour: [0, 1, 2, 3]

Problem Formulation: Phase 2

```
# Define a larger TSP problem
cities = list(range(8))
distances = {
    (0, 1): 2, (0, 2): 9, (0, 3): 10, (0, 4): 7, (0, 5): 3, (0, 6): 1, (0, 7): 6,
    (1, 2): 6, (1, 3): 4, (1, 4): 3, (1, 5): 8, (1, 6): 5, (1, 7): 2,
    (2, 3): 1, (2, 4): 8, (2, 5): 4, (2, 6): 7, (2, 7): 3,
    (3, 4): 3, (3, 5): 6, (3, 6): 2, (3, 7): 5,
    (4, 5): 1, (4, 6): 6, (4, 7): 4,
    (5, 6): 2, (5, 7): 3,
    (6, 7): 8}
```



A Larger Example

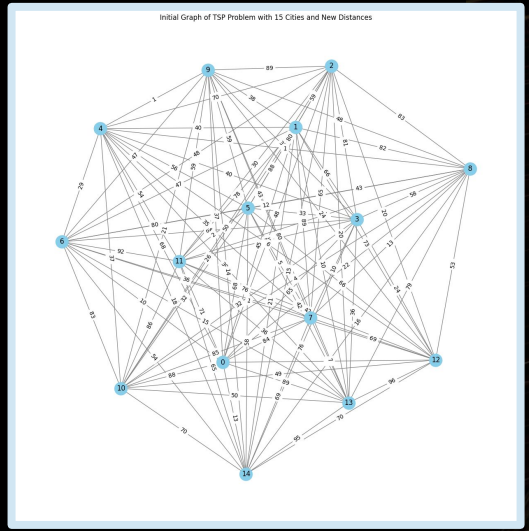


Most common bitstring: 11111111
TSP Tour: [0, 1, 2, 3, 4, 5, 6, 7]

Problem Formulation: Phase 3

The Overworked Salesman:
This is a further extension of our TSP Problem with 15 cities and a random distances generated between different pairs.

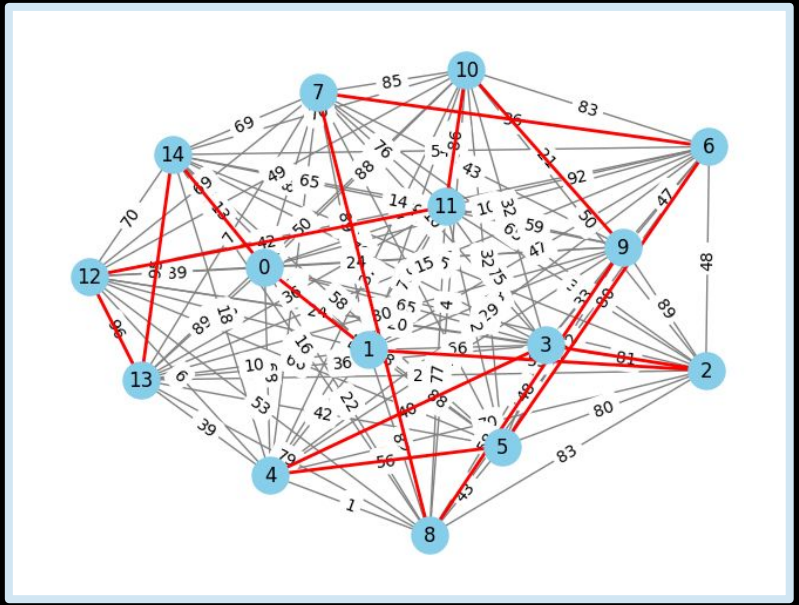
Here is the code snippet that accomplished this:




```
# Define a TSP problem with 15 cities
num_cities = 15
cities = list(range(num_cities))

# Generate random distances for the TSP problem
np.random.seed(0)
distances = {}
for i in range(num_cities):
    for j in range(i+1, num_cities):
        distances[(i, j)] = np.random.randint(1, 100)
```

Results



Most common bitstring: 111111111111111
TSP Tour: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

06

Future Directions in Quantum Computing

Where do we go from here?

Quantum vs Classical Algorithms

Classical

Algorithm: Dynamic Programming, Branch and Bound
Time Complexity: $O(n!)$
Resources: High computational power and memory
Scalability: Limited by factorial growth

Quantum

Algorithm: QAOA, Grover's Algorithm
Time Complexity: Potential polynomial speedup
Resources: Qubits for parallel state representation
Scalability: Promising for large-scale optimization

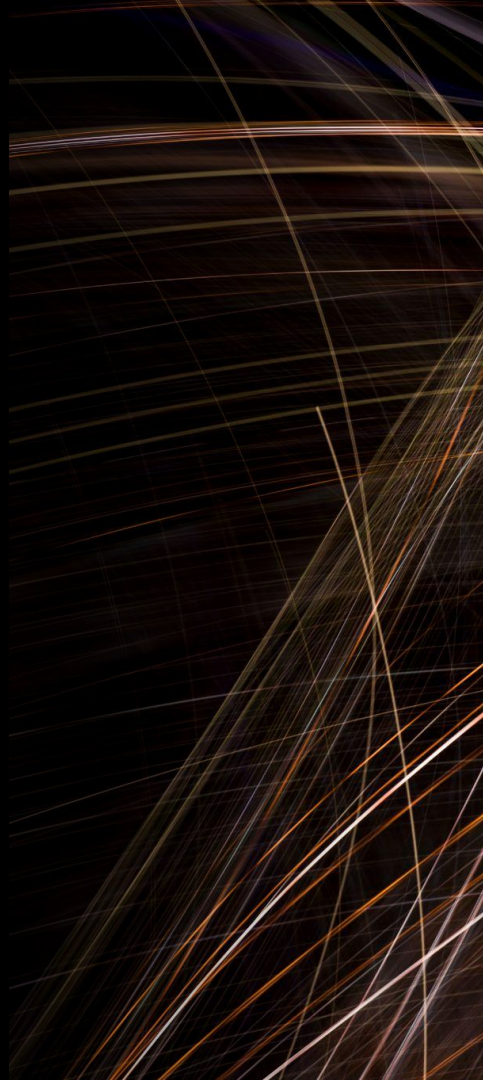
Future Directions in Quantum Computing

Advancements in Quantum Hardware:
Increasing qubit counts and reducing error rates.

Quantum Supremacy:
Achieving tasks that classical computers cannot solve in a reasonable time.

Applications:
Potential for ubiquitous application across fields of cryptography, material science, optimization problems, etc.

Research Opportunities:
Continued development of quantum algorithms and error correction techniques.



Division of Labor

Our Project

- Kai
  - Introduction of TSP, Quantum Algorithms Overview, Shor's and Grover's Algorithm, Computational Complexity
- Matthew
  - QAOA Algorithm Step Through, Computational Complexity, Demonstration Setup, Future of Quantum Computing, Quantum Optimization at a High Level, Future of Quantum Optimization

“We couldn’t build quantum computers unless the universe were quantum and computing. We can build such machines because the universe is storing and processing information in the quantum realm. When we build quantum computers, we’re hijacking that underlying computation in order to make it do things we want: little and/or/not calculations. We’re hacking into the universe.”

- Seth LLoyd

Thanks!

Matthew Torre

&

Kai Roybal

CREDITS: This presentation template was created by Slidesgo, and includes icons by Flaticon, and infographics & images by Freepik

Resources

Goldreich, O. (2008). Computational complexity: a conceptual perspective. ACM Sigact News, 39(3), 35-39.

Jain, S. (2021). Solving the traveling salesman problem on the d-wave quantum computer. Frontiers in Physics, 9, 760783.

OpenAI. (2024). Conversation with an AI assistant on solving the Traveling Salesman Problem using classical and quantum computing approaches. OpenAI ChatGPT.

Salehi, Ö., Glos, A., & Miszczak, J. A. (2022). Unconstrained binary models of the travelling salesman problem variants for quantum optimization. Quantum Information Processing, 21(2), 67.

Wong, T. G. (2022). Introduction to Classical and Quantum Computing. Rooted Grove.

Programming Citations:
https://quantumai.google/cirq/start/start

https://github.com/aubreycoffey/TSP-Quantum-Computing/blob/main/quantum_algorithms/objective.py

Resources

Hoffman, K. L., Padberg, M., & Rinaldi, G. (2013). Traveling salesman problem. *Encyclopedia of operations research and management science*, *1*, 1573-1578.

Fakhimi, R., & Validi, H. (2023). Quantum Approximate Optimization Algorithm (QAOA).

Stackpole, Beth (2024). Quantum Computing: What leaders need to know now.

UC Berkeley (2024). Quantum algorithms.

Arxiv (2023). Quantum Optimization: Potential, Challenges, and the Path Forward.