Quantum Optimization for Energy-Efficient Route-Finding

Nathan Pacey, Cherilyn Christen, Lindon Zymberi

Presentation Outline

- Data Centers & Energy usage
- Route Optimization Algorithms
- Example: UPS
- Traveling Salesman Problem
- Algorithm Performance

- Classical Approach
- Qunatum Approach
- Time Complexity
- Energy Consumption
- SDG’s
- Questions

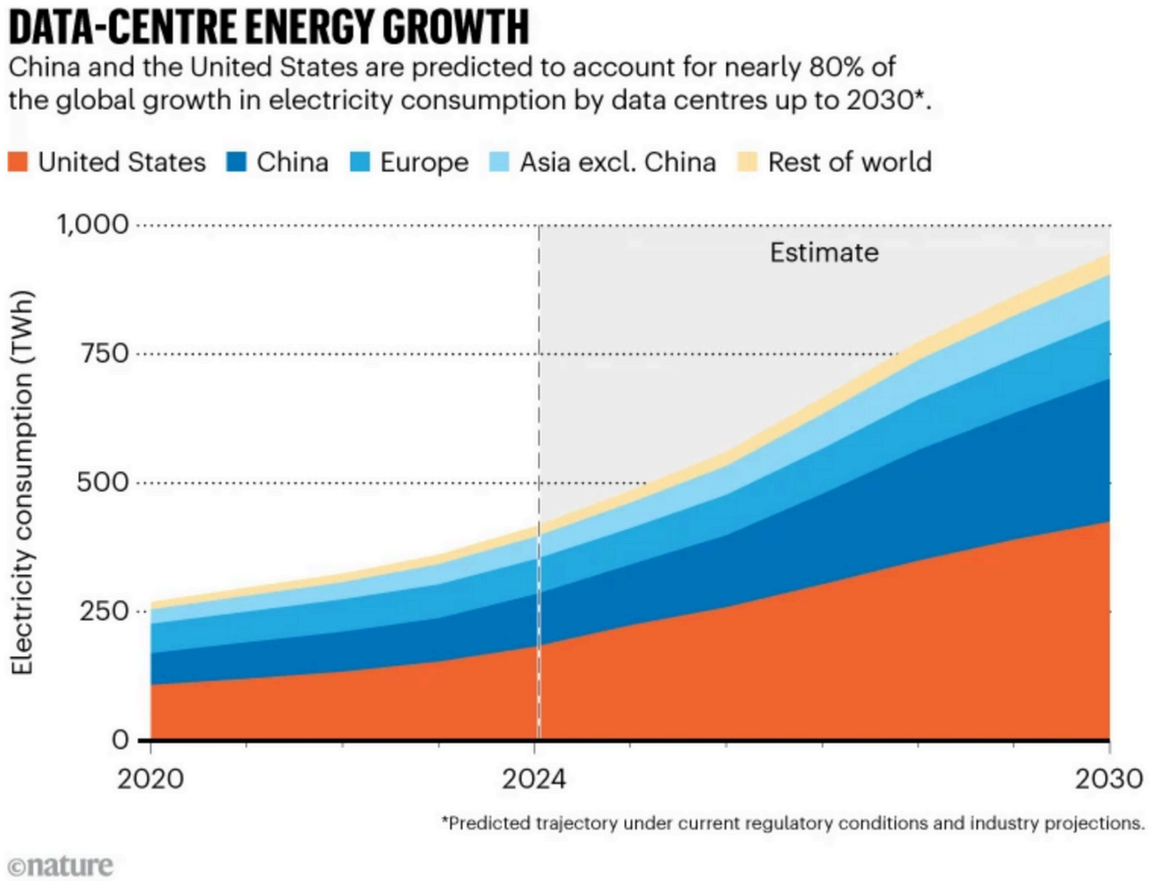
Data Centers

- Physical facilities that house computer servers and networking equipment.
- Power everyday apps: Google Maps, Uber, Netflix, AI models, financial systems, national infrastructure, etc.
- Data centers consume 1–2% of global electricity (IEA 2022).
- Projected to rise to 6–8% by 2030.
- Servers & CPUs/GPUs: 40–60% of Energy Use

Stargate Proejct - Open AI

Let’s put that in perspective...

246 households!

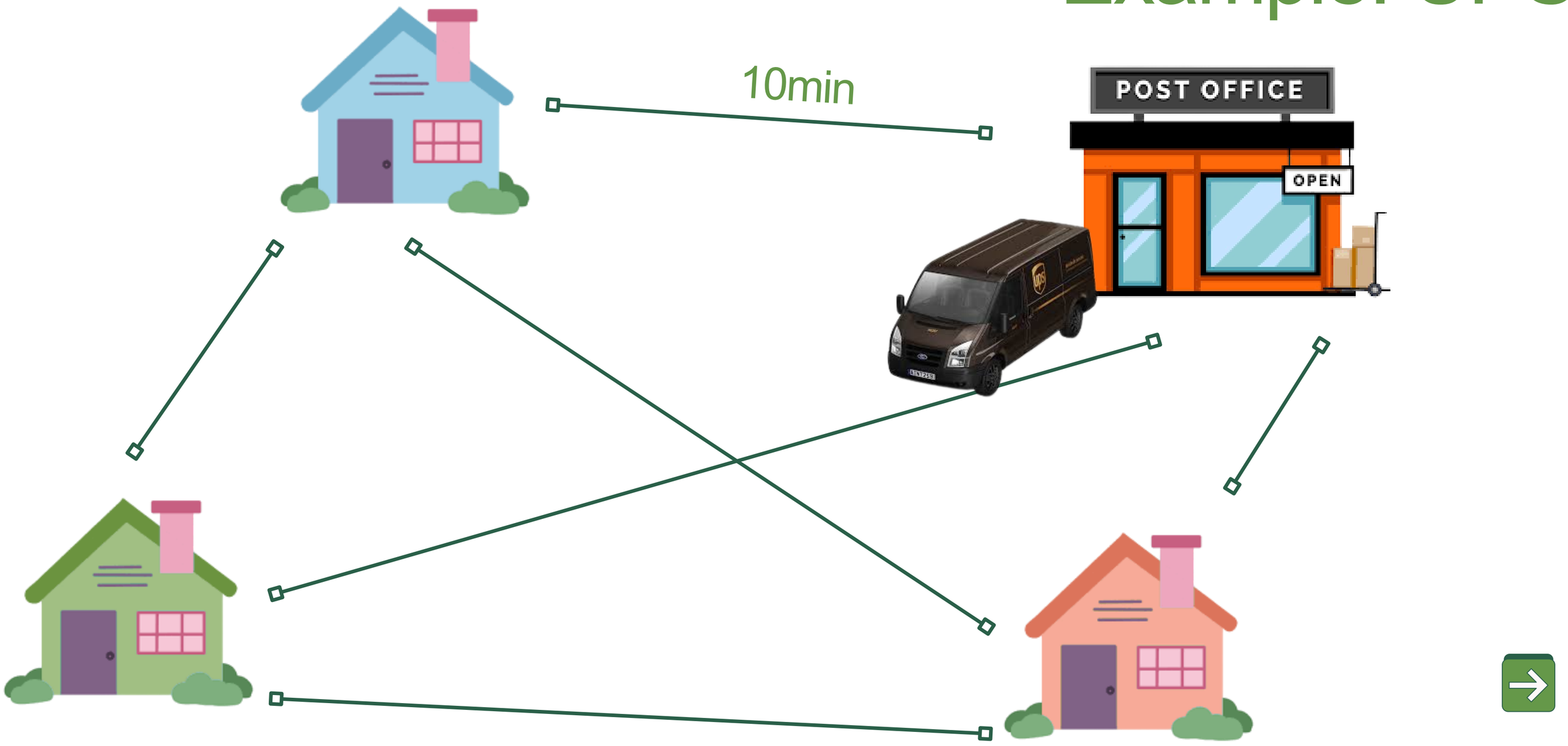


Source: IEA. CC BY 4.0

Algorithms

- whether it's a search query, a delivery route, or a train scheduling problem — we rely on algorithms.
- depends on complexity of the task
- Use case: UPS

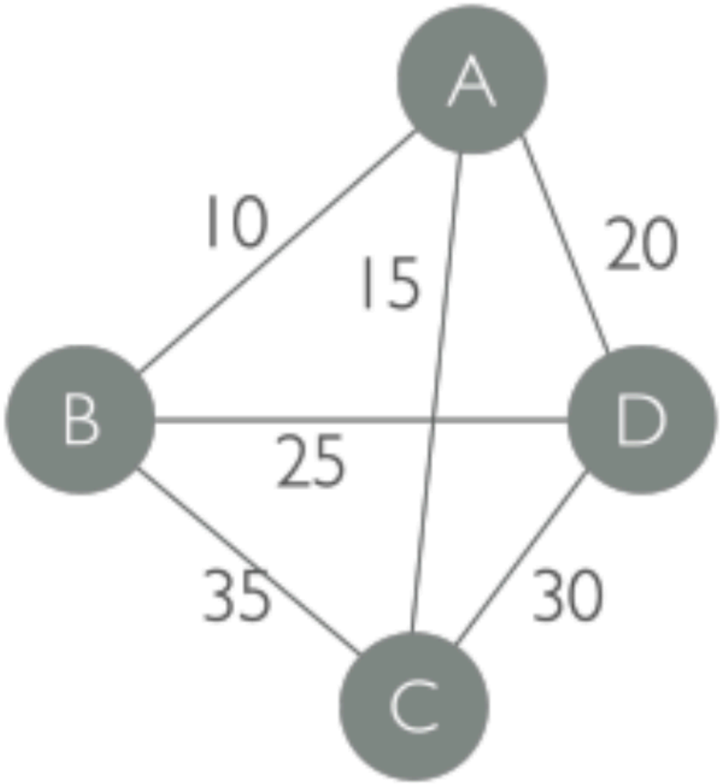
Example: UPS



Example: UPS

1) start and end
at the post office (A)

2) Visit every House
exactly once



Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization. Given a list of n cities and a distance function $d(i, j)$ that gives the cost of travel between any pair of cities i and j, the goal is to find the shortest possible route that:

- Starts and ends at the same city
- Visits each other city exactly once

Traveling Salesman Problem

Mathematical Formulation

Let the set of cities be $V = \{0, 1, 2, \ldots, n-1\}$. A tour is a permutation π of the cities such that:

$$\pi : \{0, 1, \ldots, n-1\} \to V$$

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \pi(1) & \pi(2) & \cdots & \pi(n) \end{pmatrix}$$

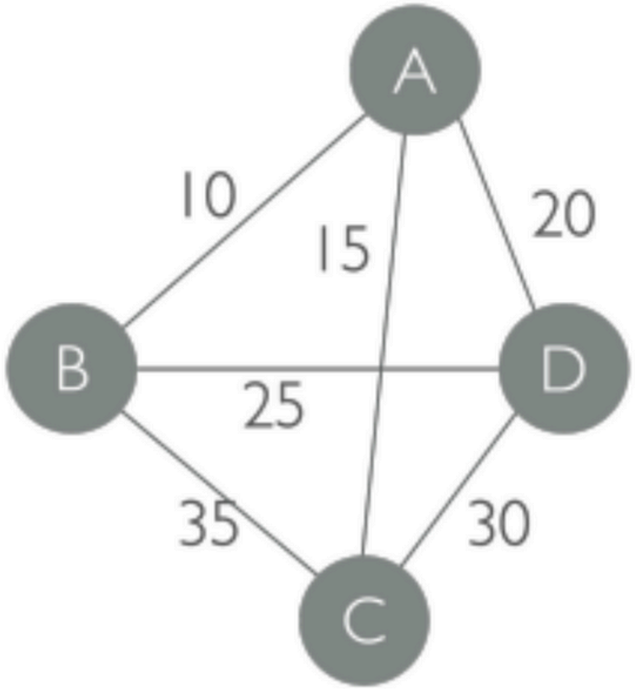
The cost of a tour is:

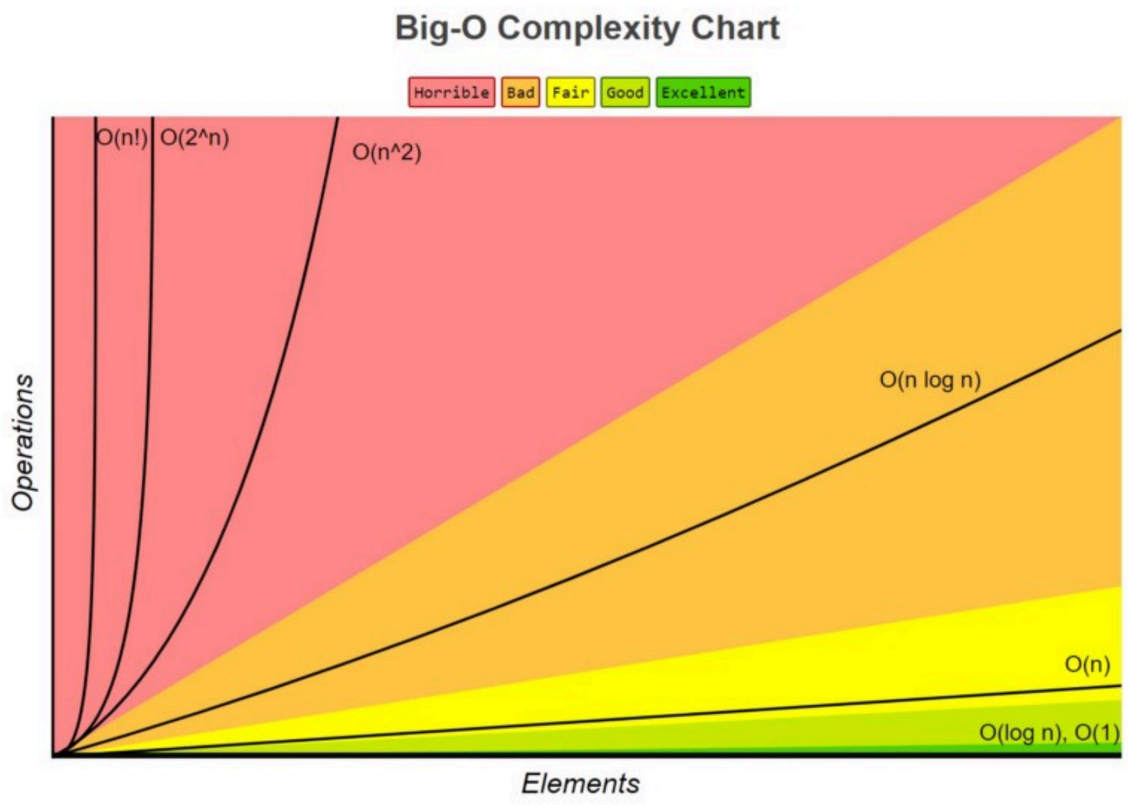
$$\text{Cost}(\pi) = \sum_{i=0}^{n-2} d(\pi(i), \pi(i+1)) + d(\pi(n-1), \pi(0))$$

The TSP asks for a permutation π that minimizes this cost.

Solve the TSP

- TSP’s complexity grows exponentially with the number of cities/households.
- Distinguish Algorithms by Efficiency
- Since TSP is hard to solve exactly:
  - Exact vs. heuristic methods
- trade off: accuracy vs. resources





Big O Notation

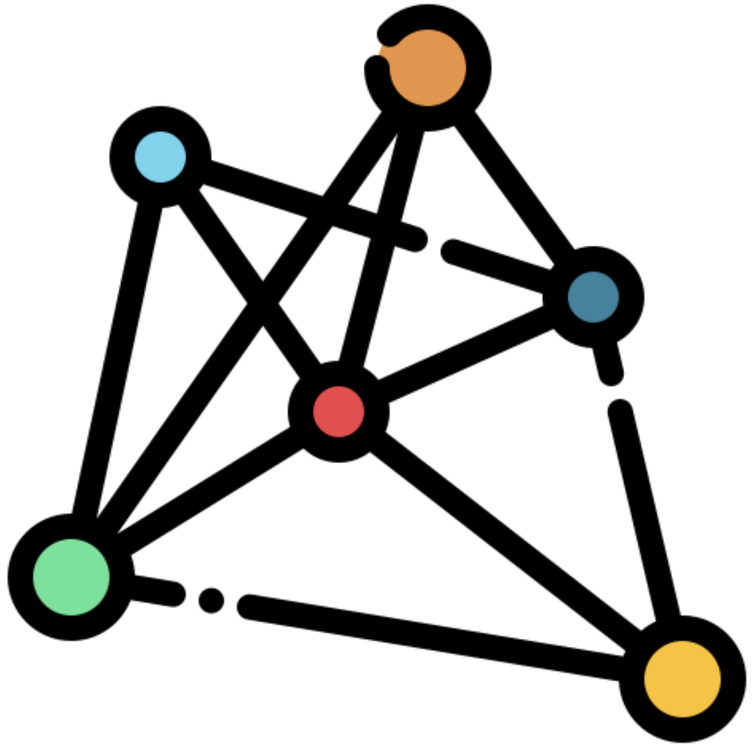
- Number of steps (operations) in relation to input size.
- compare algorithm efficiency

- Helps predict scalability

Each step a CPU takes uses energy So the more steps, the more energy consumed.

Goal: Optimize Algorithm Efficiency

Quantum computing offers new ways to process information, potentially solving hard problems more efficiently...

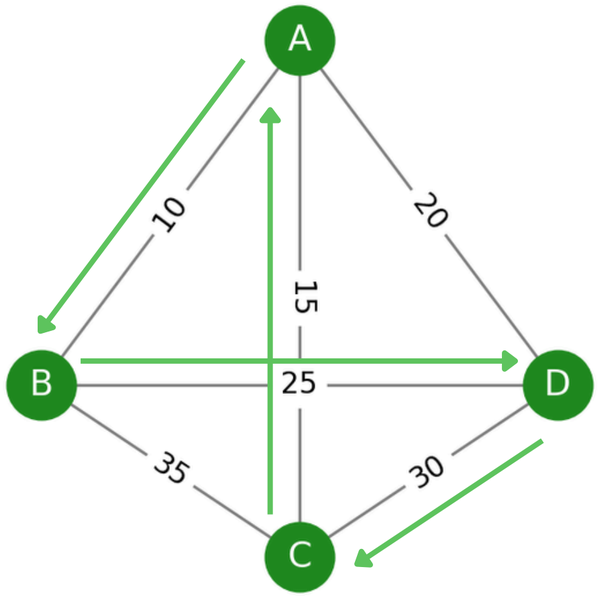


CLASSICAL APPROACH

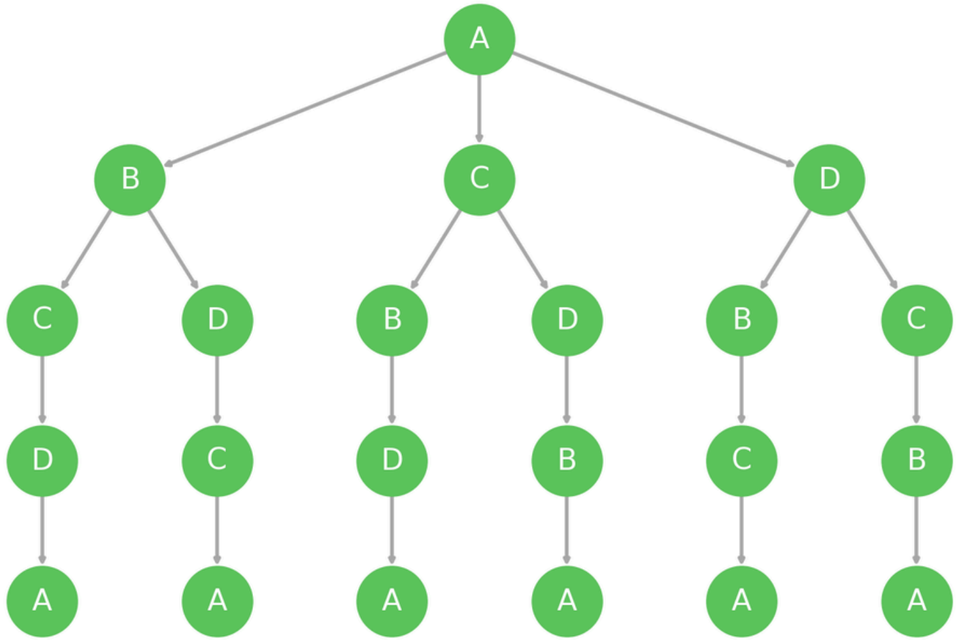
Classical Approach I: Brute Force

- Brute Force explores all possible paths exhaustively
- For N cities, this means (N–1)! permutations
- Guaranteed to find the optimal route
- Becomes impractical quickly exponential time complexity: O(n!)
- Memory usage is low only the current best path is stored

Traveling Salesman Graph (4 Cities)



Brute Force Decision Tree for 4-City TSP



Classical Approach II: Held-Karp (Dynamic Programming)

Let $C(S, j)$ be the length of the shortest path that starts at city 0, visits all cities in subset $S \subseteq V$ (with $0 \in S$), and ends at city $j \in S$, where $j \neq 0$. Then:

$$C(\{0\}, 0) = 0$$

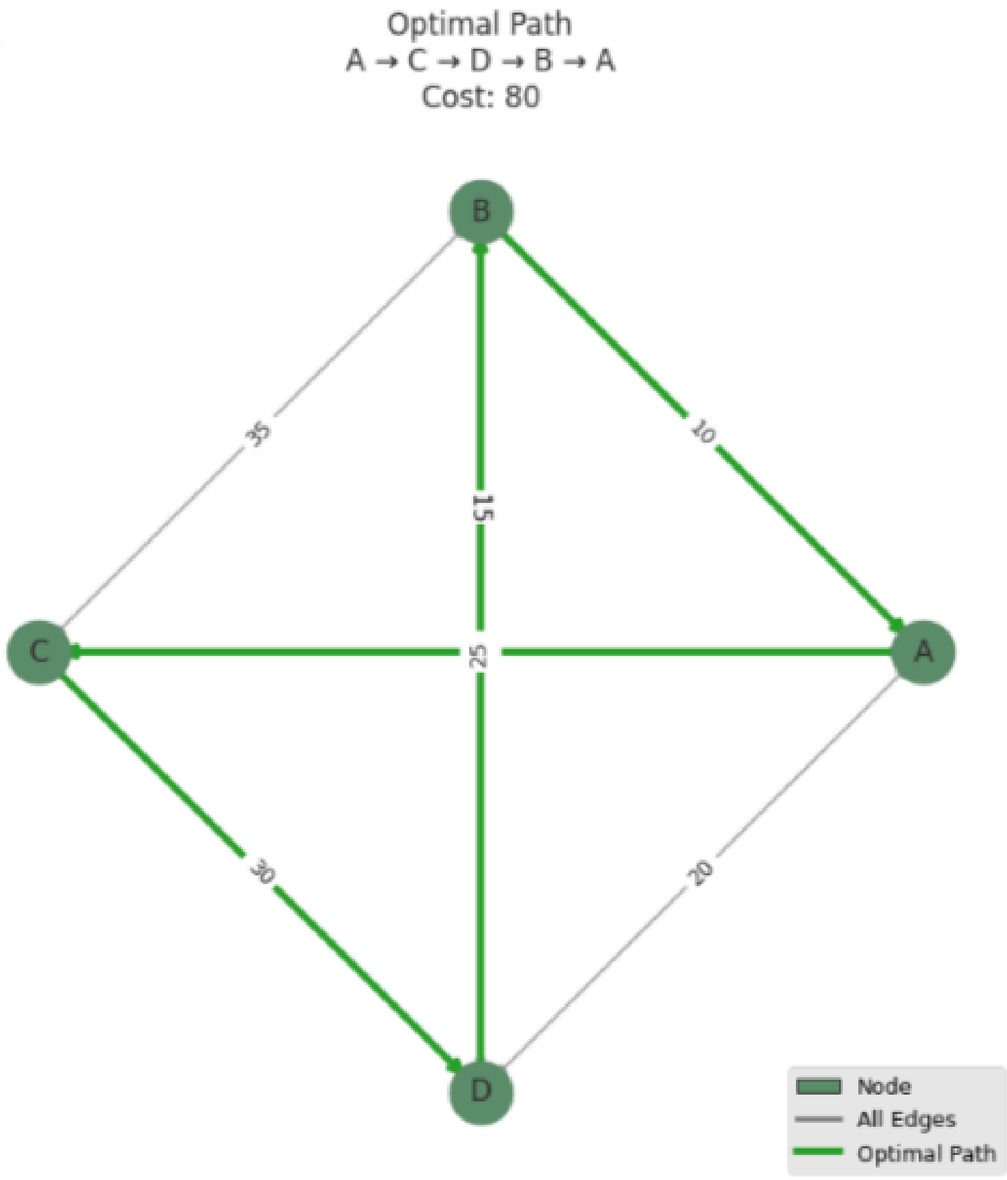
$$C(S, j) = \min_{\substack{i \in S \\ i \neq j}} \left[C(S \setminus \{j\}, i) + d(i, j) \right], \quad \text{for } |S| > 1$$

This recurrence builds up solutions by extending smaller tours by one city at a time [1]. Once all $C(S, j)$ values are known, the final solution is computed as:

$$\min_{\substack{j \in V \\ j \neq 0}} \left[C(V, j) + d(j, 0) \right]$$

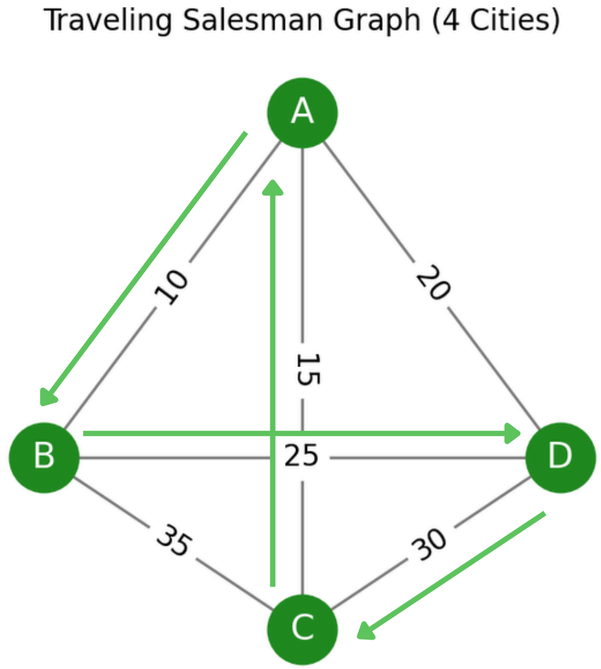
Classical Approach II: Held-Karp (Dynamic Programming)

- Uses dynamic programming to avoid redundant calculations
- Solves subproblems once and stores results
- Significantly reduces the number of operations
- Time complexity: $O(N^2 2^N)$– better than brute force
- Memory trade-off: requires more space for caching
- Still exponential, but more efficient for moderately sized problems



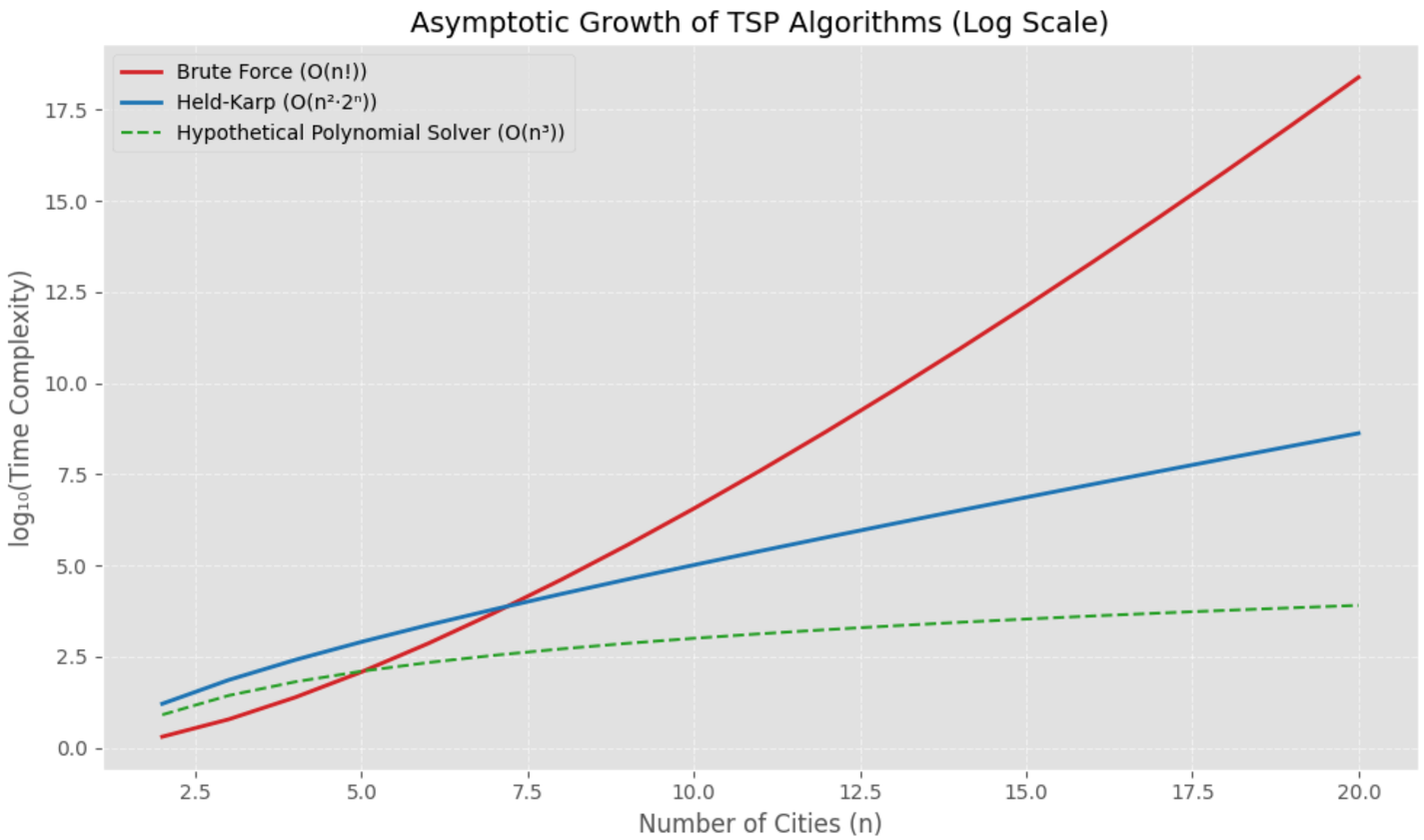
Classical Comparison

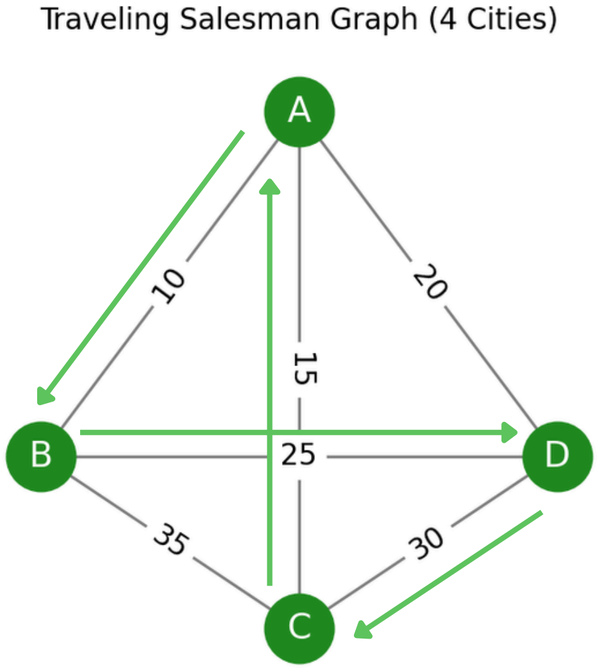
- Brute Force: checks all possible permutations (n!)
  – Impractical beyond ~10 cities
- Held-Karp: dynamic programming with subproblem reuse
  – Reduces time from O(n!) to $O(n^2 \cdot 2^n)$
- Still exponential in size — infeasible for large graphs
- Best solution depends on graph size and structure
- Motivates approximate or heuristic approaches for large n
- Can quantum algorithms (like QAOA) help?



Algorithm	Time Complexity	Space Complexity	Runtime	Memory Usage
Brute Force	O(n!)	O(n)	~0.6 ms	0.625 KB
Held-Karp	$O(n^2 \cdot 2^n)$	$O(n \cdot 2^n)$	~0.2 ms	1.281 kB

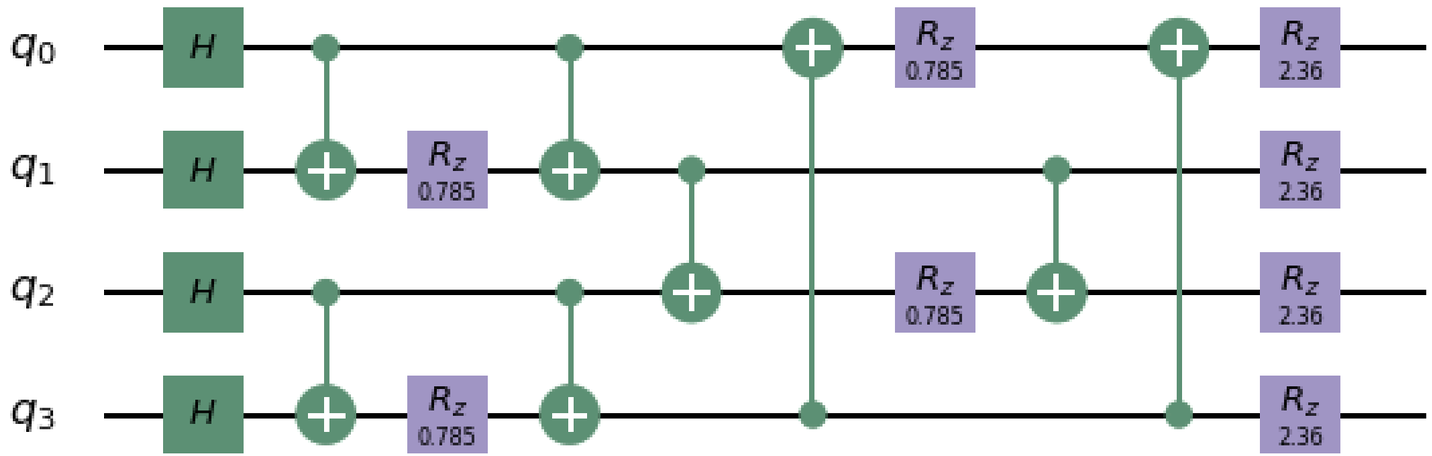
Classical Comparison



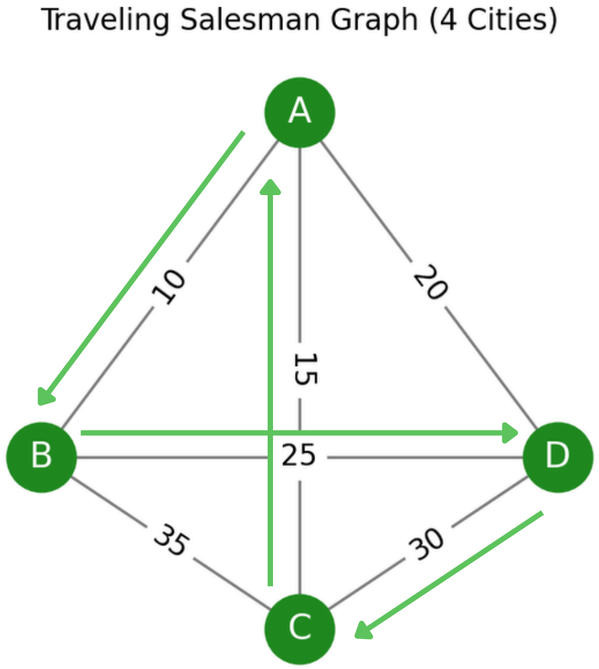


Algorithm	Time Complexity	Space Complexity	Pros	Cons
Brute Force	**O(n!)**	**O(n)**	**Simple, exact**	**Infeasible for n > 10**
Held-Karp	**$O(n^2 \cdot 2^n)$**	**$O(n \cdot 2^n)$**	**Exact, DP optimization**	**Still exponential**

QUANTUM APPROACH



Quantum Approximate Optimization Algorithm (QAOA) for TSP



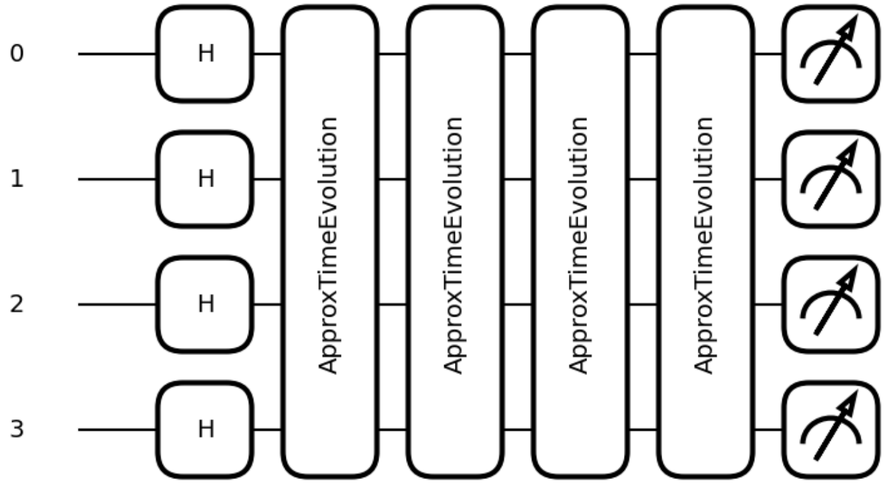
- QAOA is a hybrid quantum-classical algorithm for combinatorial optimization

- TSP encoded into a quantum cost Hamiltonian:
  - Distance matrix → problem constraints
  - Mixer Hamiltonian allows state transitions

- Circuit alternates layers of:
  1. Cost unitary (encodes objective)
  2. Mixer unitary (explores solution space)

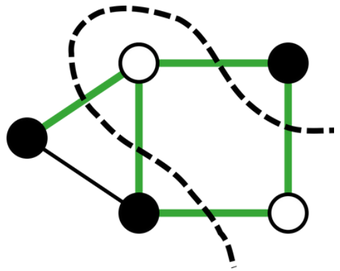
- Parameters (γ, β) are optimized via classical optimizer

- Final quantum state → measured to extract likely optimal paths



Farhi et al. - A Quantum Approximate Optimization Algorithm ↩️

Quantum Approximate Optimization Algorithm



Given an undirected weighted graph $G = (V, E)$, the **MaxCut problem** seeks a partition of the nodes into two sets such that the total weight of edges crossing the cut is maximized.

Each node $i \in V$ is represented by a qubit in state $|0\rangle$ or $|1\rangle$, indicating which side of the partition it's in.

For a graph with edge weights w_{ij}, the **MaxCut cost Hamiltonian** is:

$$e^{-i\gamma H_C} = \prod_{(i,j)\in E} e^{-i\gamma w_{ij} \frac{1-Z_iZ_j}{2}} \qquad H_C = \sum_{(i,j)\in E} w_{ij} \cdot \frac{1 - Z_iZ_j}{2}$$

This rewards cuts (i.e., when $Z_iZ_j = -1$) by increasing the cost.

The **mixer Hamiltonian** is a simple transverse-field term encouraging exploration:

$$e^{-i\beta H_M} = \prod_{i\in V} e^{-i\beta X_i} \qquad H_M = \sum_{i\in V} X_i$$

This is implemented via single-qubit $RX(2\beta)$ rotations.

For p layers, the full QAOA circuit prepares the variational state:

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

- $|+\rangle^{\otimes n}$ is the uniform superposition over all bitstrings.
- The parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$ are optimized to maximize the expectation of the cost Hamiltonian.

Farhi et al.

QAOA Layers and Circuit

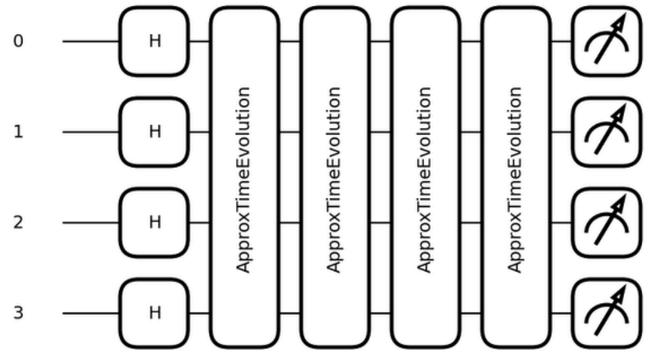
```
def qaoa_layer(gamma, alpha): ...
def circuit(params): ...
```

- **Layer Definition**: Defines a function to apply cost and mixer Hamiltonians to the qubits.
- **Circuit Definition**: Constructs the QAOA circuit with alternating layers of Hamiltonians.

Optimization Process

```
optimizer = qml.GradientDescentOptimizer()
params = np.random.rand(2 * num_nodes)

for i in range(steps):
    params = optimizer.step(cost_function, params)
```



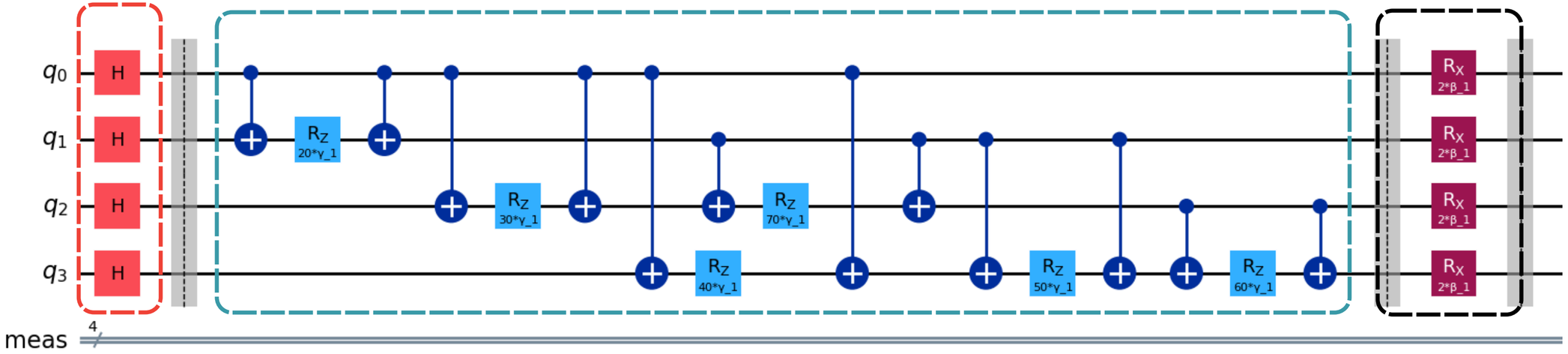
- **Optimizer Choice**: Gradient descent is used for optimizing the parameters in the QAOA circuit.
- **Parameter Initialization and Optimization**: Parameters are randomly initialized and iteratively optimized to minimize the cost function.

- Probability Measurement: After optimization, the quantum circuit's output state probabilities are measured.
- Bitstring to Path Conversion: The most probable bitstring is translated back into a path representing a potential solution to the TSP.

Super-Position Encoding

QAOA Cycle 1: Cost Hamiltonian

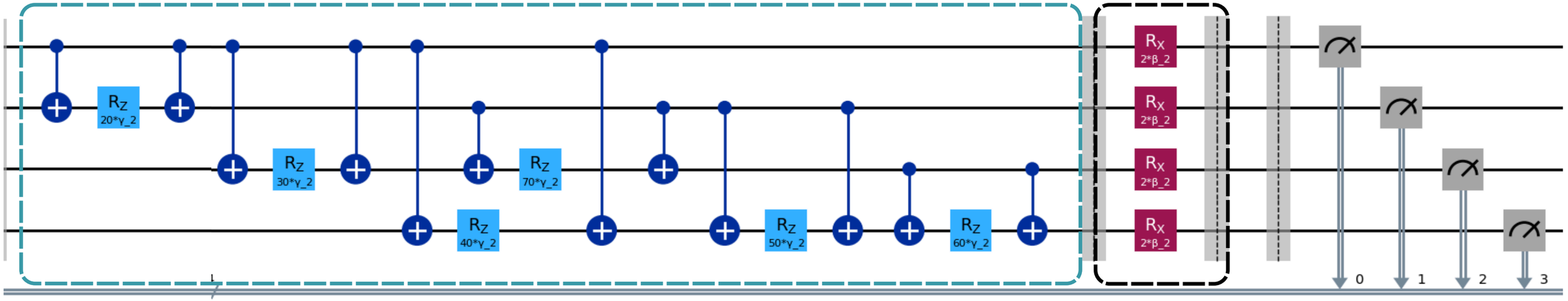
Mixer Hamiltonian



QAOA Cycle 2: Cost Hamiltonian

Mixer Hamiltonian

Measurements



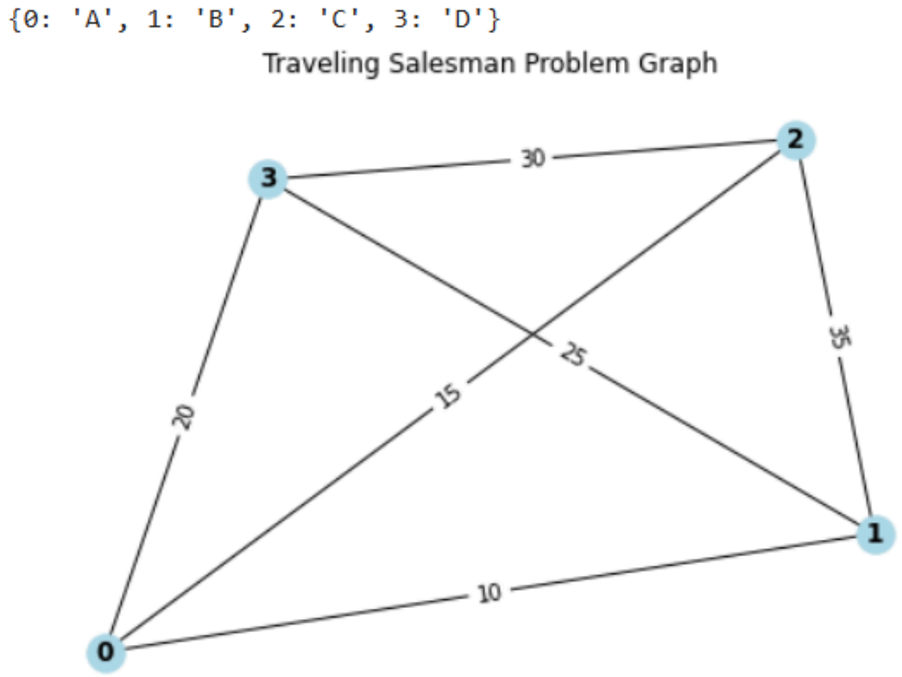
Farhi et al. - A Quantum Approximate Optimization Algorithm ↩

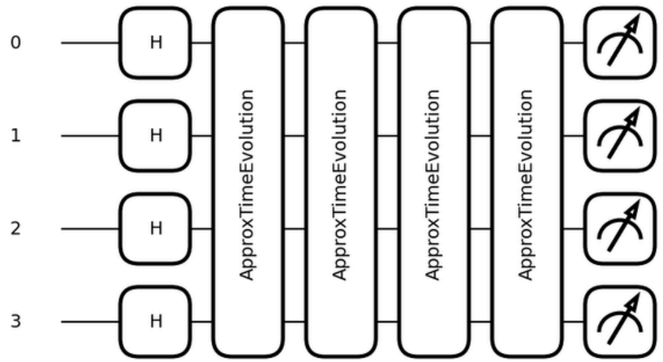
QAOA for TSP

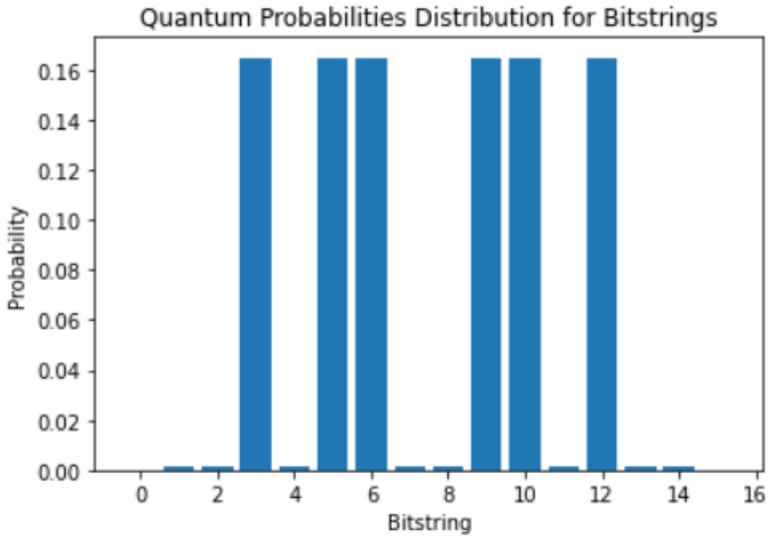
Most probable bitstring: 3
Binary or State Representation: [0, 0, 1, 1]
Where the digits correspond to the first two path choices: [A,B,C,D]

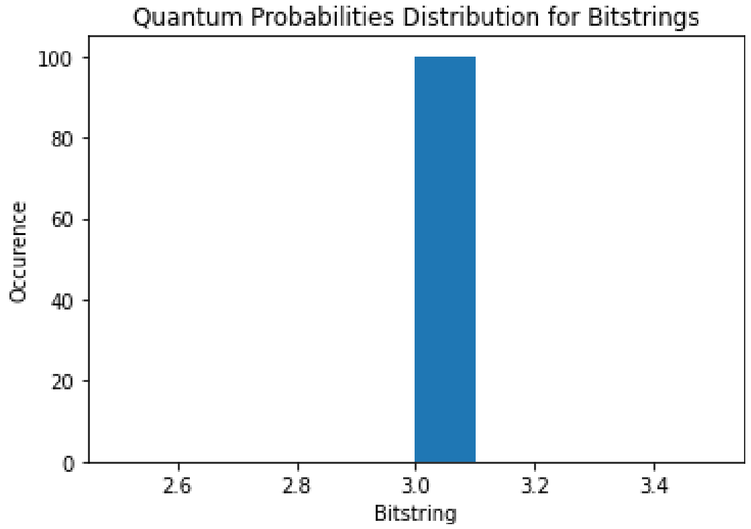
The shortest path starting from city 'A' is: A -> C -> D -> B -> A
The total distance of the shortest path is: 80 units.

Runtime of the code: ~8.90 seconds
Memory usage: ~193.78 MB





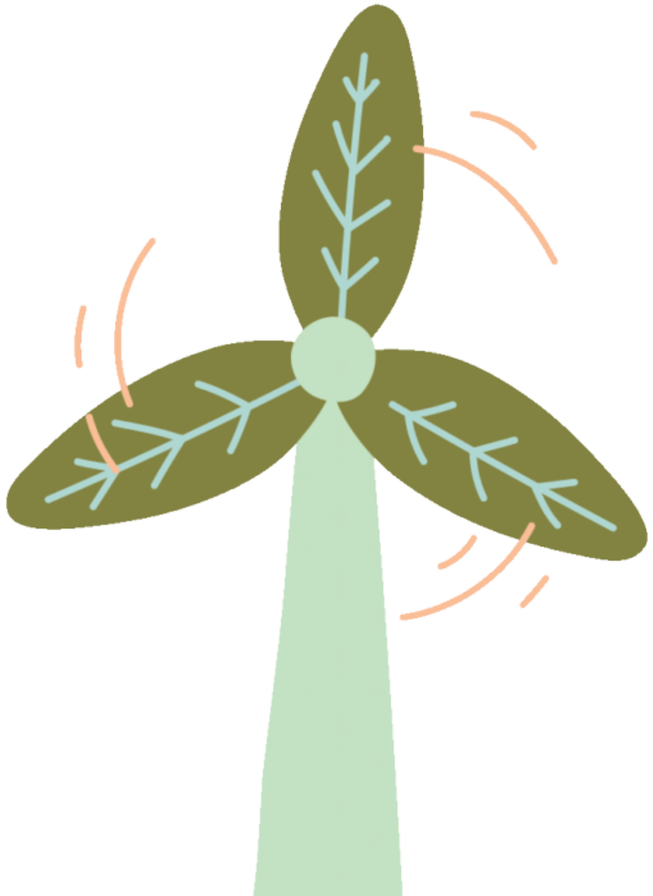


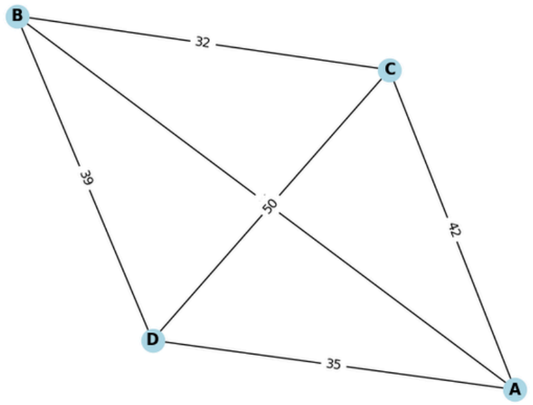


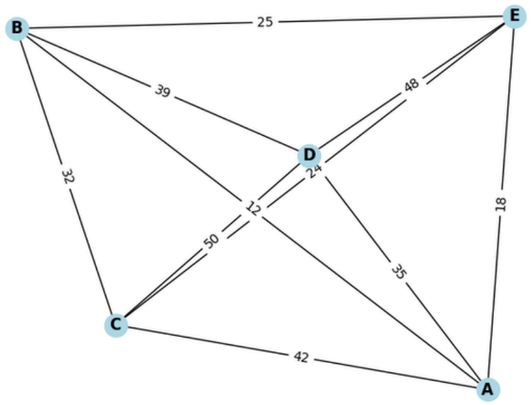
Estimate Energy Use

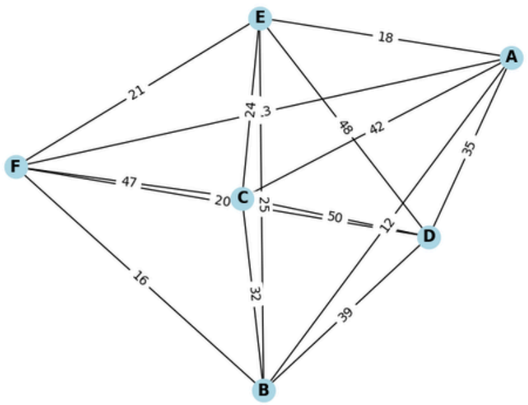
Classical Algorithm: $O(N^2 2^N)$

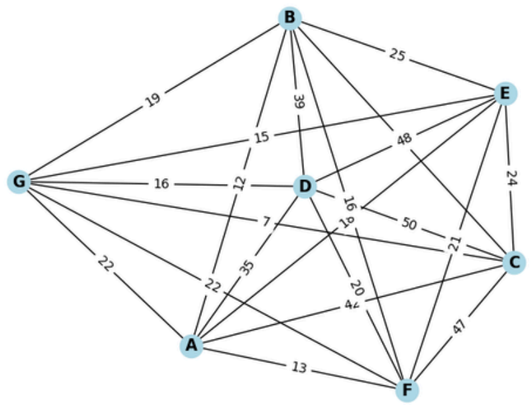
Quantum Algorithm: $O(N)$

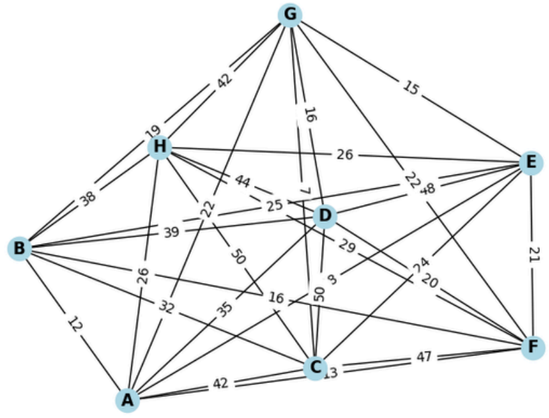


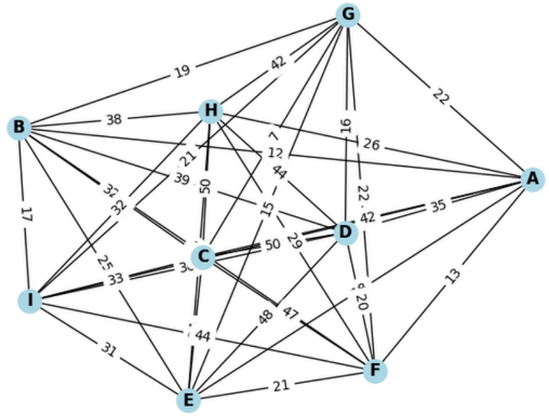

B
C
D
A
32
39
50
42
35

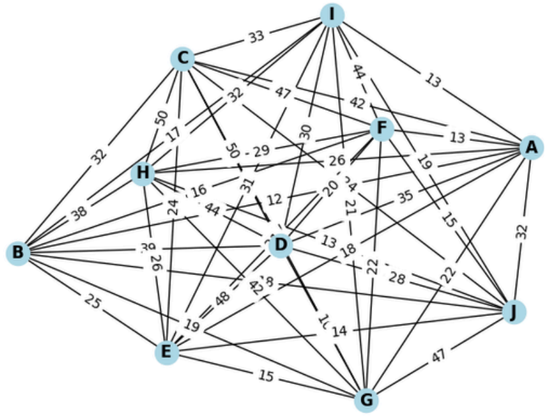

B
E
D
C
A
25
39
48
24
32
12
50
35
18
42

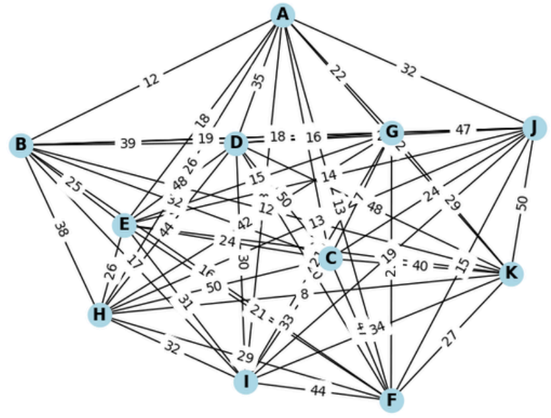

E
A
F
C
D
B
18
21
24
3
48
42
35
47
20
25
50
12
16
32
39

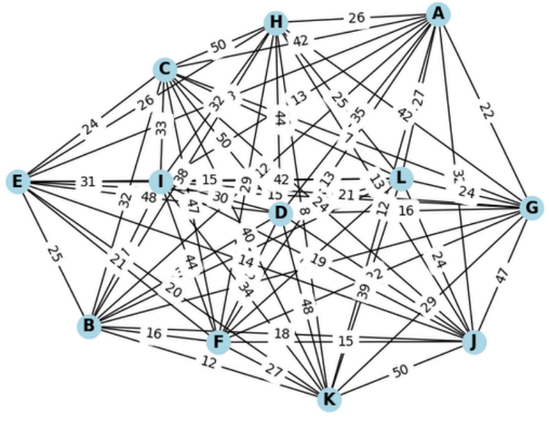

B
E
G
D
C
A
F
25
19
39
15
48
24
16
12
7
16
50
21
22
35
22
20
42
13
47

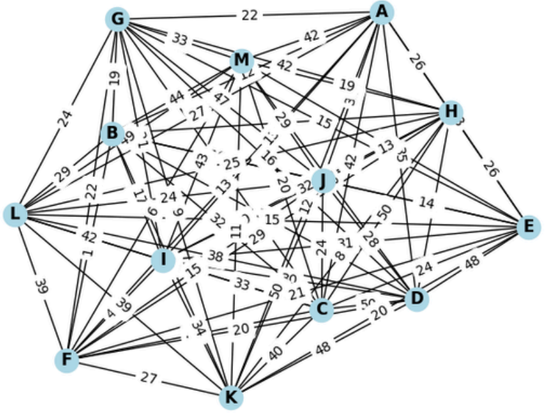

G
H
E
D
B
C
A
F
42
15
19
16
26
38
44
22
48
25
39
7
50
29
74
20
21
26
32
35
16
50
12
13
47
42

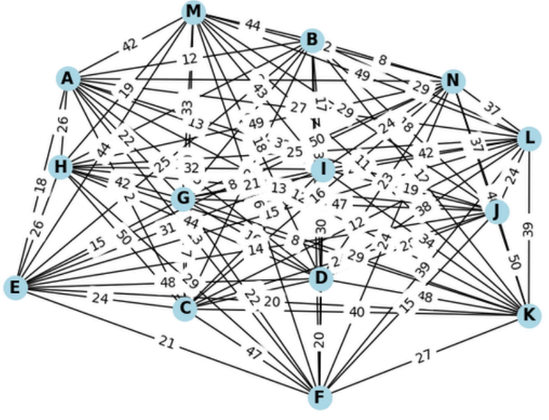

G
H
B
A
I
C
D
F
E
19
42
22
38
16
26
12
21
39
44
32
50
15
17
22
42
35
25
33
29
48
47
20
13
31
21

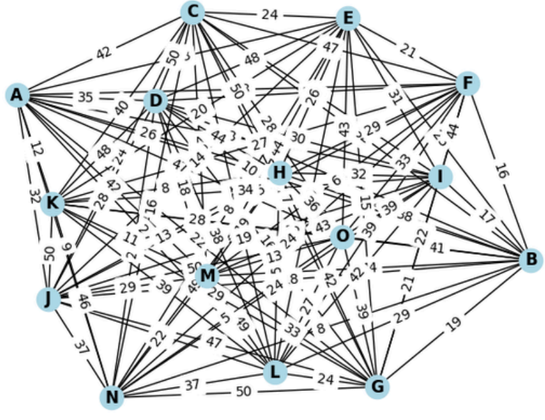

I
C
F
A
H
B
D
E
G
J
33
44
13
32
47
42
50
17
30
29
26
19
38
16
31
12
20
24
35
15
44
21
24
32
25
48
19
14
47
15


A
B
D
G
J
E
C
K
H
I
F
12
35
22
32
18
16
39
19
47
48
26
25
38
44
42
24
29
50
17
30
15
40
27
31
50
8
21
33
34
44


H
A
C
E
I
L
G
D
B
F
J
K
26
50
42
24
33
32
44
13
35
27
22
31
38
15
12
29
48
47
30
21
16
25
40
14
19
24
34
39
18
12
27
50


G
A
M
H
B
L
J
E
I
C
D
F
K
22
33
42
26
19
24
29
44
27
15
13
25
16
20
32
50
14
39
1
38
30
21
48
12
11
6
31
40
18
36
34
50


M
B
A
N
L
H
G
I
J
E
D
C
K
F
44
42
12
2
8
49
29
37
26
18
33
43
27
17
50
25
32
13
41
24
22
6
15
47
38
21
30
39
48
31
14
34
20
40


C
E
A
F
D
H
I
K
O
B
J
M
L
G
N
24
42
21
50
48
47
35
40
26
12
43
29
44
32
33
16
17
39
22
41
18
46
37
27
34
10
8
6
15
38
36
19
24
5
4
42
21
29
19
50

Energy Consumption
Brute Force

- Check all (n−1)! possible permutations (fixing the starting city).
- Each check takes 1 µs (1e-6 s) – very optimistic
- CPU runs at 200 W = 0.2 kW.

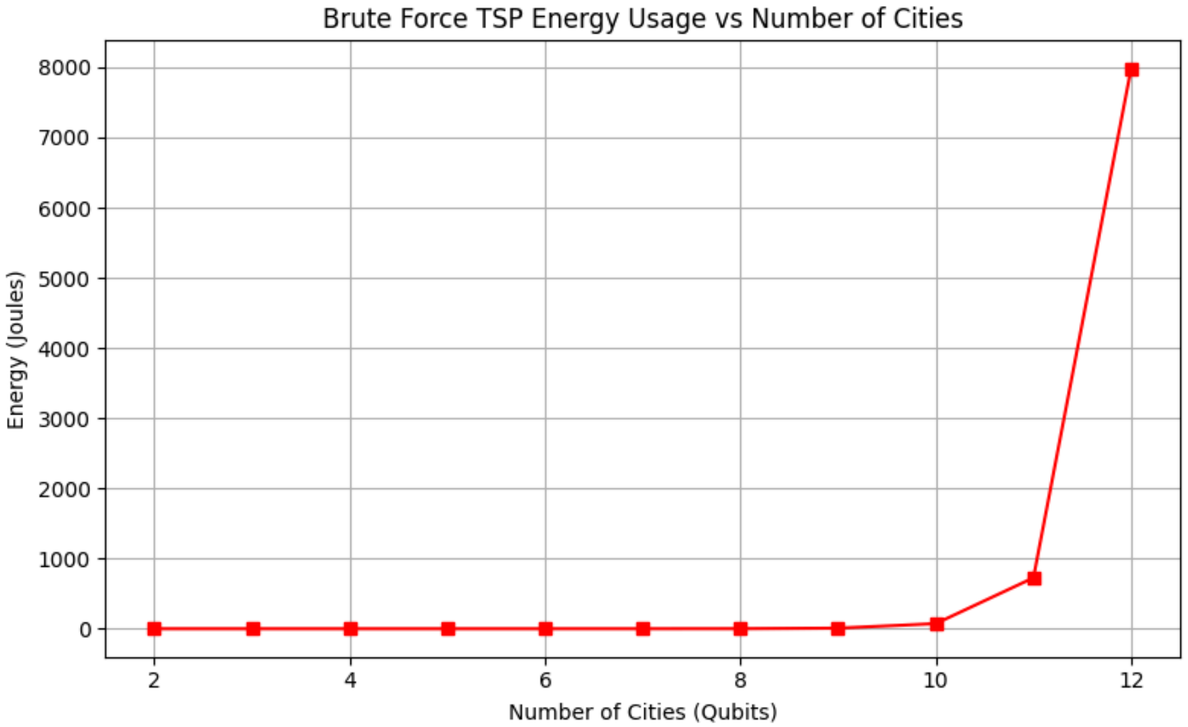
$$\text{Energy}_{\text{brute}}(n) = \frac{(n-1)! \cdot 1\mu s}{3600} \times 0.2\,\text{kWh}$$

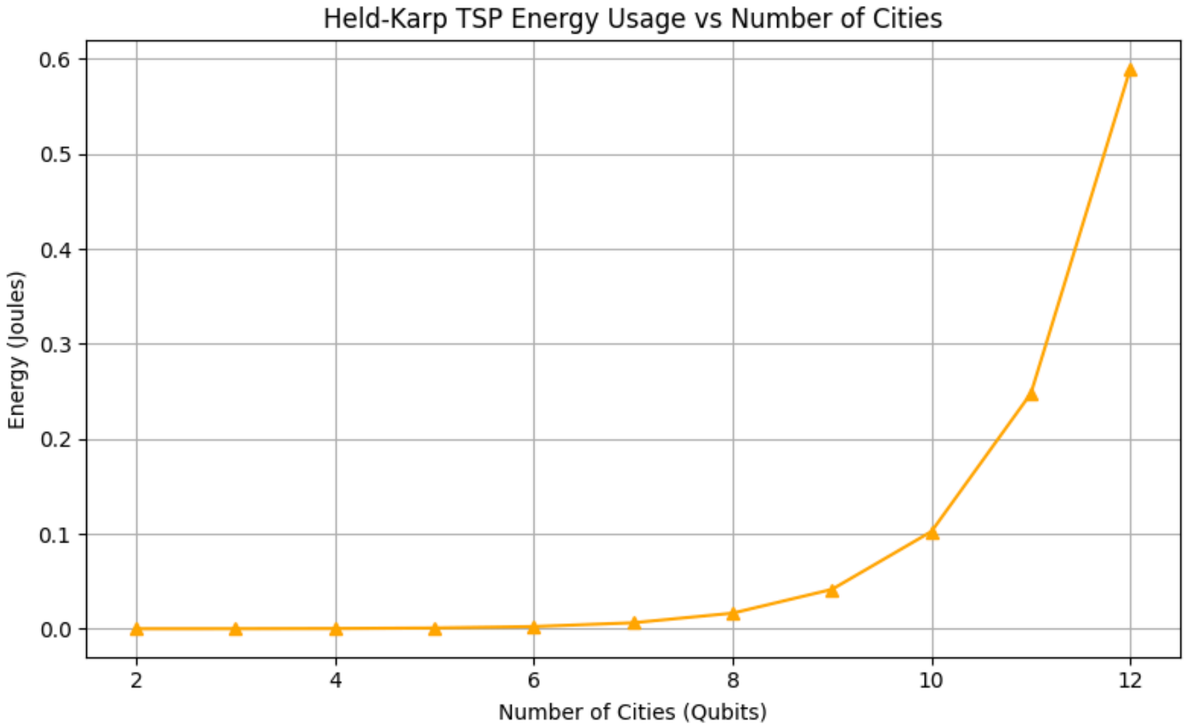
Energy Consumption
Held-Karp

- Time complexity: $O(n^2 \cdot 2^n)$
- Each operation takes 5 ns (5e-9 s)
- CPU runs at 200 W = 0.2 kW.

$$\text{Energy}_{\text{HK}}(n) = \frac{n^2 \cdot 2^n \cdot 5\text{ns}}{3600} \times 0.2\,\text{kWh}$$

Energy Consumption
Classical Algorithms

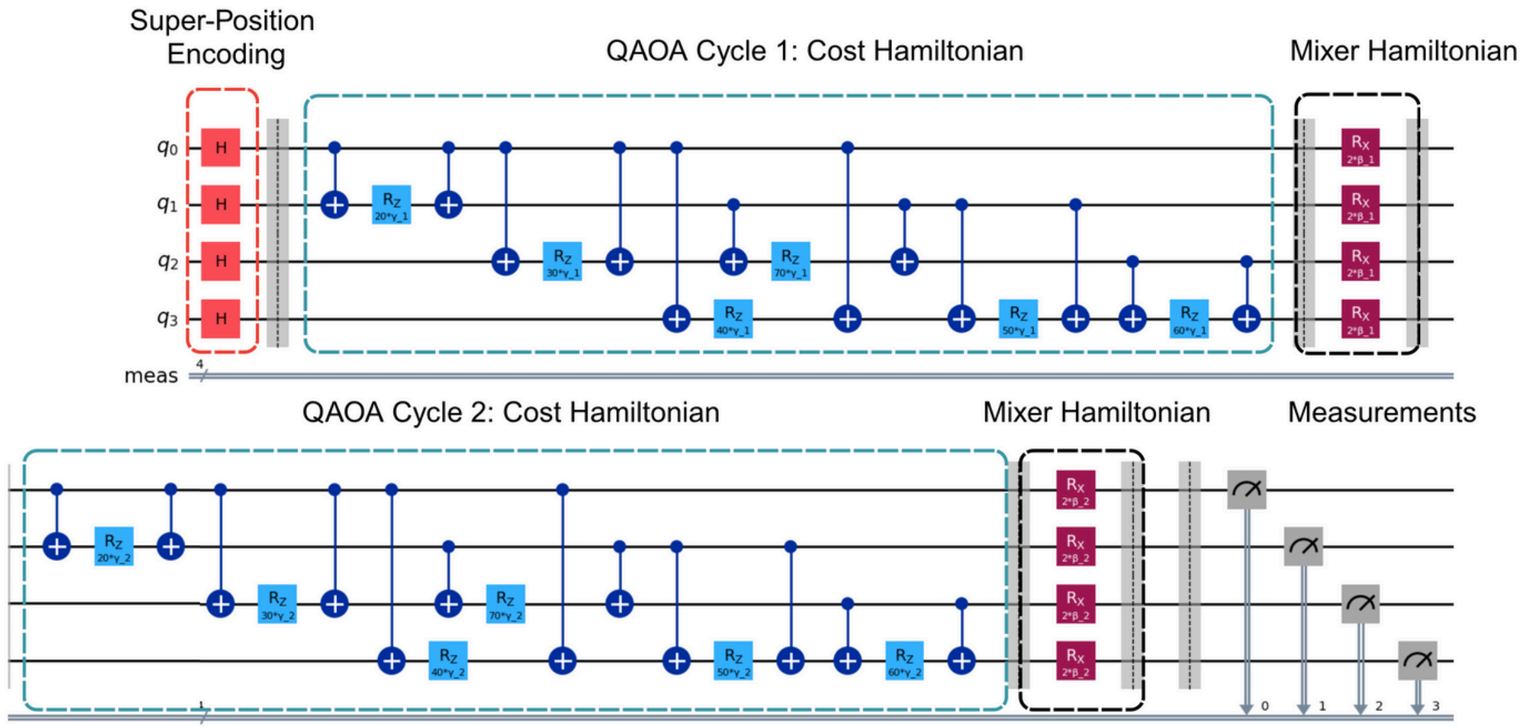




Energy Consumption
QAOA

Number of gates per cycle:

H gates	4
Measurement gates	4
Rx gates	4
Rz gates	6
CNOT gates	12



Energy Consumption
QAOA

Scaling to n nodes:

H gates	n
Measurement gates	n
Rx gates	n
Rz gates	1/2 n(n-1)
CNOT gates	n(n-1)

p: number of cycles

$$N_{1q} = p \cdot (2n + \frac{n(n-1)}{2}) \rightarrow \text{total 1q gates (H, Rx, Rz)}$$

$$N_{2q} = p \cdot n(n-1) \rightarrow \text{total 2q gates (CNOTs)}$$

$$N_{meas} = n$$

Energy Consumption
QAOA

Gate and measurement times from the IBM Sherbrooke:

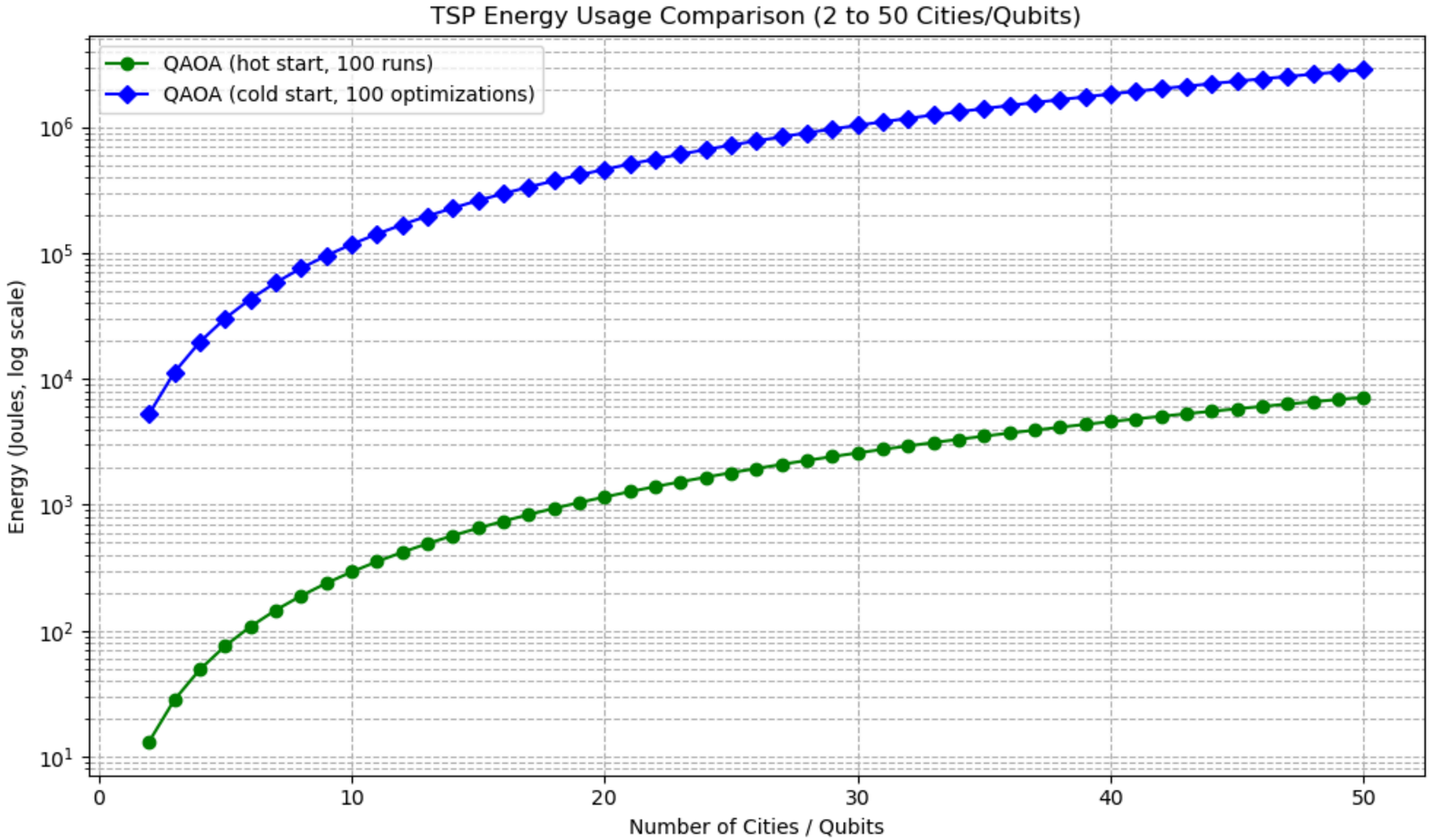
T_{1q} ≈ 5.69e-08 s
T_{2q} ≈ 5.33e-07 s
T_{meas} ≈ 1.216e-06 s

IBM power draw: $P_{QPU} = 25\,\mathrm{kW}$

$$E_{hot} = \left(\frac{T_{1q} \cdot N_{1q} + T_{2q} \cdot N_{2q} + T_{meas} \cdot N_{meas}}{3600} \right) \cdot \mathrm{runs} \cdot P_{QPU}$$

$$E_{cold} = \left(\frac{T_{1q} \cdot N_{1q} + T_{2q} \cdot N_{2q} + T_{meas} \cdot N_{meas}}{3600} \right) \cdot (\mathrm{steps} \cdot \mathrm{shots} \cdot \mathrm{repeats}) \cdot P_{QPU}$$

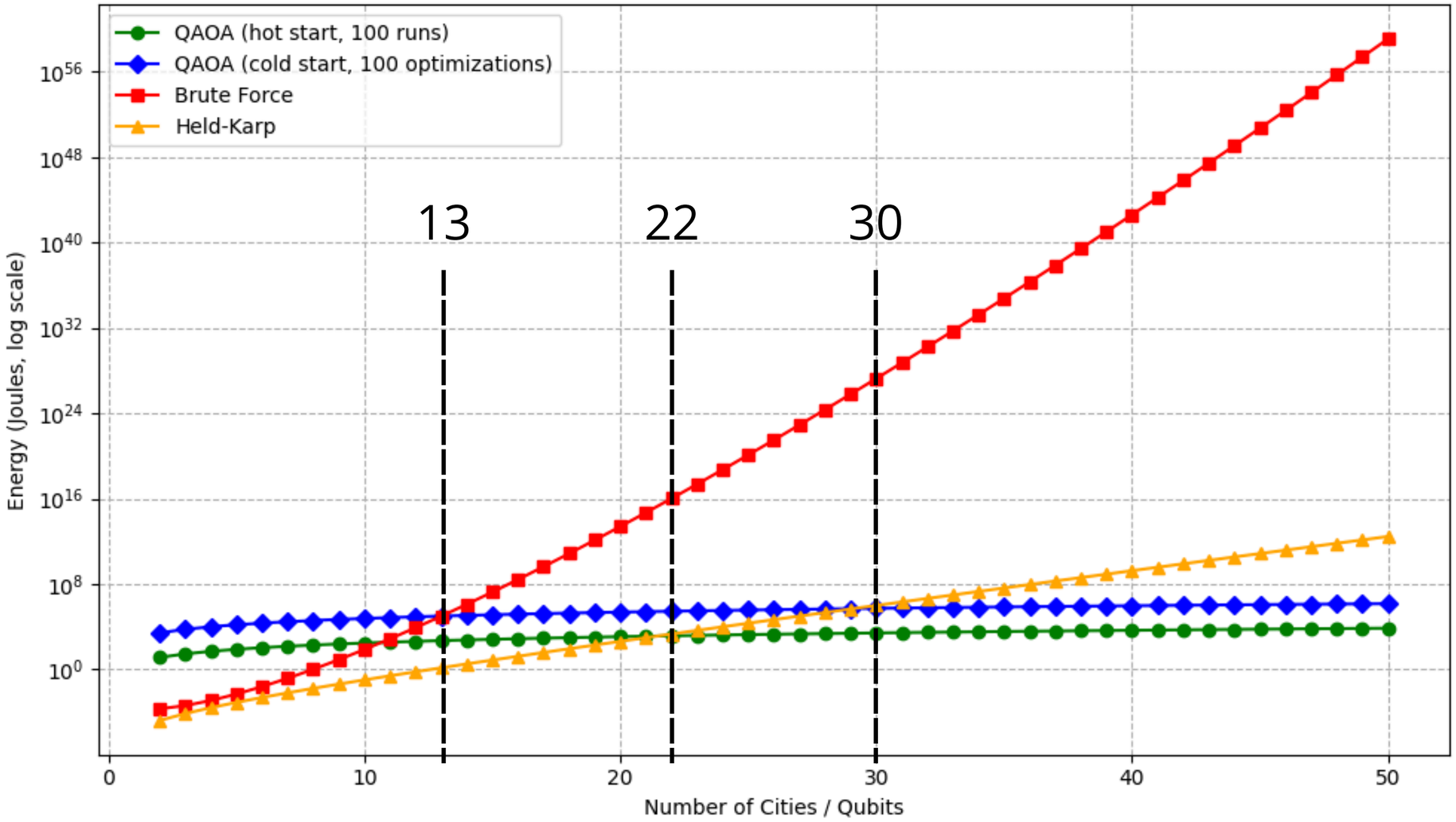
Energy Consumption
QAOA: Hot vs. Cold Start



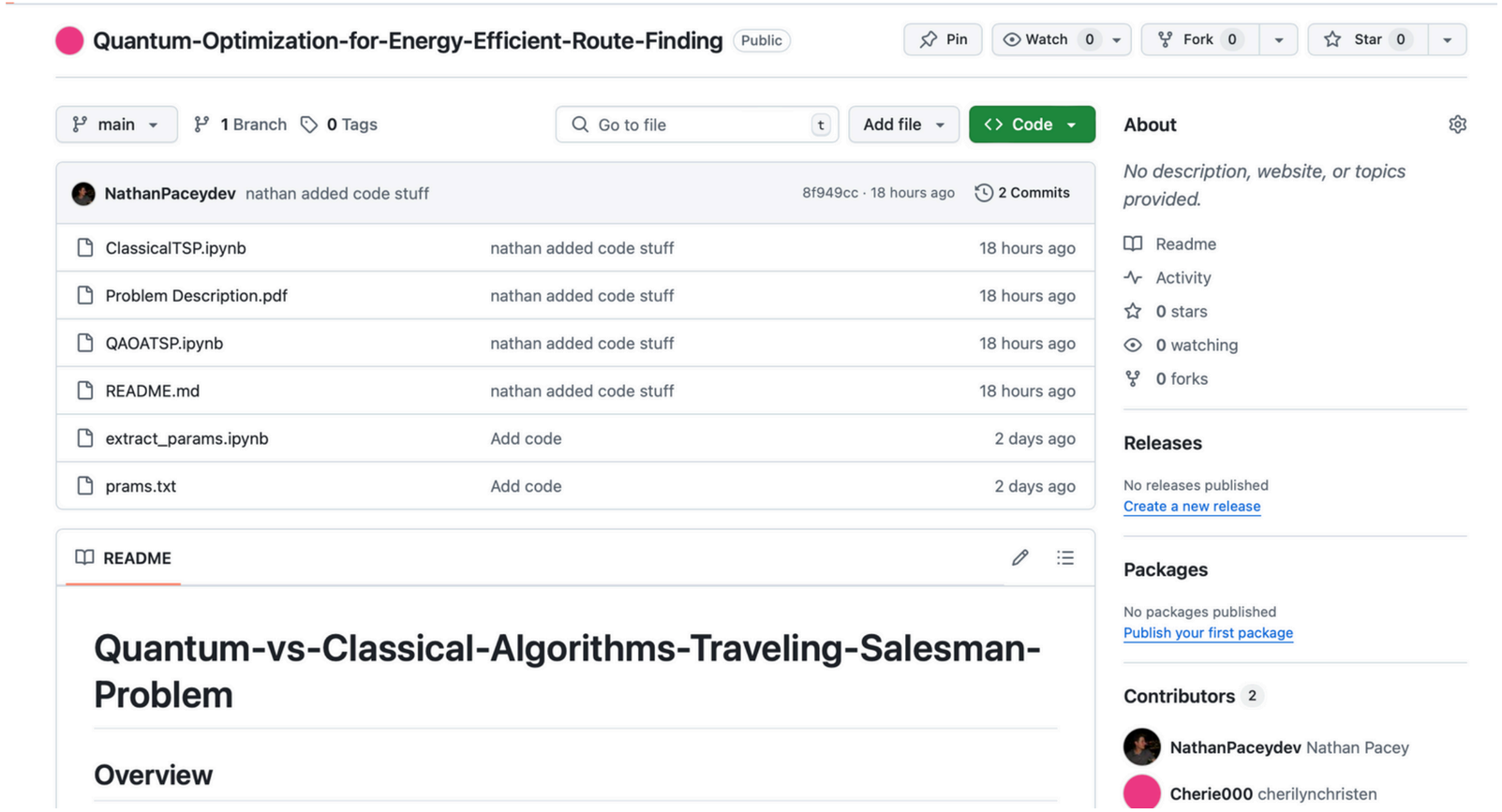
runs = 100
steps = 1
shots = 200
repeats = 200

TSP Energy Usage Comparison

2 to 50 Nodes



The code



https://github.com/Cherie000/Quantum-Optimization-for-Energy-Efficient-Route-Finding.git

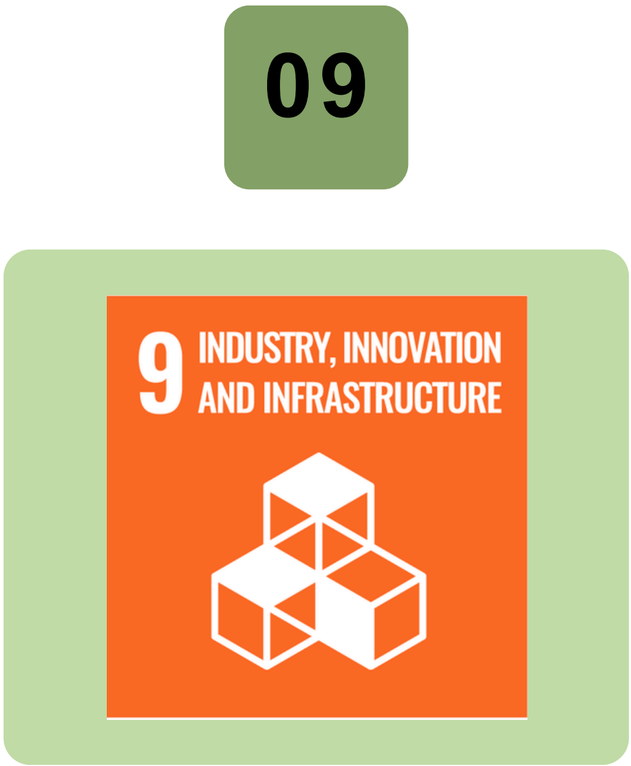
Sustainable Development Goals

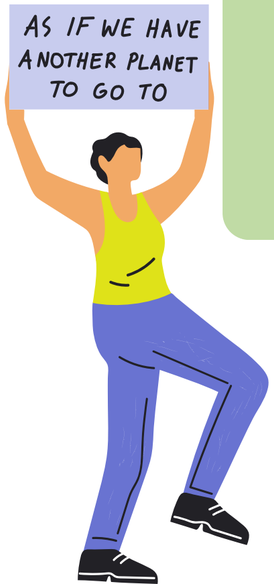
THE GOALS

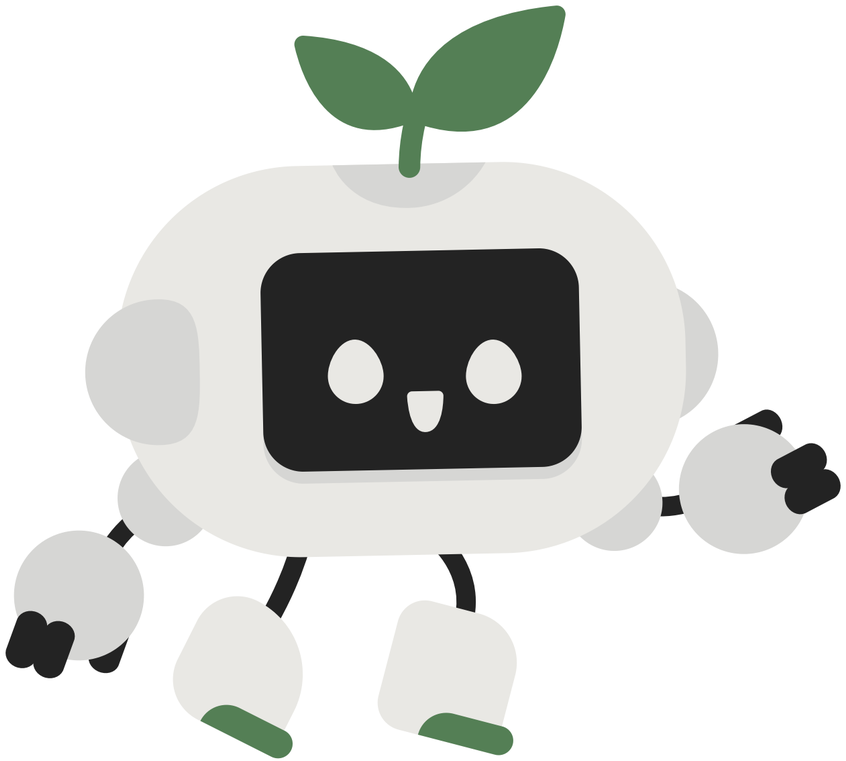
09

11

12







THANK YOU!

Data Centers & Energy use

Efficient, Demand Flexible Networked Appliances EDNA

Pascal QAOA Report

Farhi et al. - A Quantum Approximate Optimization Algorithm ↩️

GitHub

https://github.com/Cherie000/Quantum-Optimization-for-Energy-Efficient-Route-Finding.git

References

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," arXiv:1411.4028, 2014.

[2] IBM Quantum, "Qiskit Runtime Backend Properties," IBM Sherbrooke, Accessed via Qiskit API, 2025.

[3] J. Koomey, "Growth in Data Center Electricity Use 2005 to 2010," Analytics Press, 2011. `https://www.analyticspress.com/datacenters.html`

[4] International Energy Agency, "Data Centres and Data Transmission Networks," 2022. `https://www.iea.org/reports/data-centres-and-data-transmission-networks`

[5] UPS, "ORION: On-Road Integrated Optimization and Navigation," Technical Overview, 2016.

[6] T. H. Cormen et al., *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.

[7] M. Held and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *SIAM Journal*, vol. 10, no. 1, pp. 196–210, 1962.

[8] IBM Quantum Sustainability Report, 2023. Internal estimate based on IBM Q System One design.

[9] United Nations, "Transforming our world: the 2030 Agenda for Sustainable Development," United Nations General Assembly, 2015. `https://sdgs.un.org/goals`