QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM

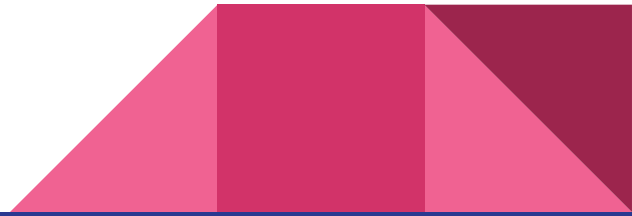
A Quantum Approximate Optimization Algorithm by Edward Farhi, Jeffrey Goldstone, Sam Gutmann, https://doi.org/10.48550/arXiv.1411.4028

Qiskit Summer School 2021

Reviewed and Presented by Dheeraj NVS

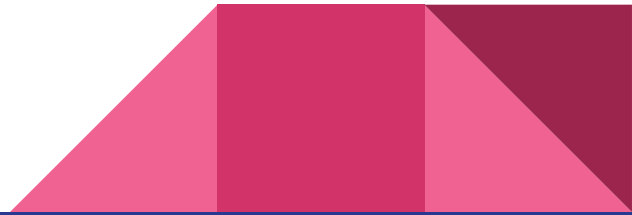
Outline

- Motivation
- Background
- Technical Approach/ Method
- Results
- Pros & Cons
- Conclusion



Motivation

- Traditional algorithms struggle with certain NP-hard problems like the Max Cut, often used in network design, circuit layout, and data science. QAOA, leveraging the principles of quantum mechanics, can potentially solve these problems more efficiently.
- QAOA is not limited to a single type of problem. Its framework is adaptable to various optimization problems, making it a versatile tool.
- This is a hybrid algorithm, so it has scope in near term devices.



Background

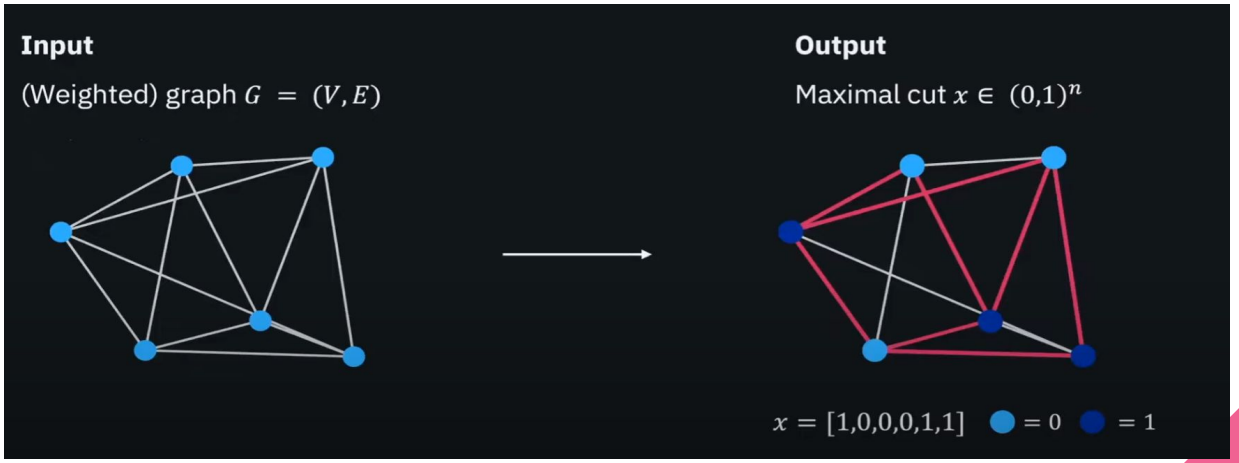
- This algorithm can be thought of a special case of the VQE.
- This algorithm can specifically solve QUBO(Quadratic Unconstrained Binary Optimization) problems.
- The variational model consists of the following steps:
  - Choose an Ansatz or trail state parameterized by Θ.
  - Vary parameters Θ to minimize the energy level.

$$E(\theta) = \langle \psi(\theta) | H | \psi(\theta) \rangle$$

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E(\theta)$$

MaxCut Problem

- A graph is a set of vertices V connected by (weighted) edges E.
- A cut of the graph is the partition of the vertices into two disjoint sets and the weight of the cut is the cumulative weight of the edges connecting the vertices from the two sets.



MaxCut to QUBO and QUBO to Hamiltonian

MaxCut

Weight matrix

$$W = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 4 \\ 2 & 3 & 0 & 1 \\ 3 & 4 & 1 & 0 \end{pmatrix}$$

Cost function

$$C(x) = \sum_{i,j=1}^{n} W_{ij} x_i (1 - x_j)$$

QUBO

QUBO matrix and vector

$$c_i = \sum_{j=1}^{n} W_{ij} \qquad Q_{ij} = -W_{ij}$$

Cost function

$$C(x) = \sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i=1}^{n} c_i x_i = x^T Q x + c^T x$$

Goal: Find Hamiltonian operator H_C that encodes cost function $C(x)$

$$H_C|x\rangle = C(x)|x\rangle$$

QUBO cost function

$$C(x) = \sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i=1}^{n} c_i x_i$$

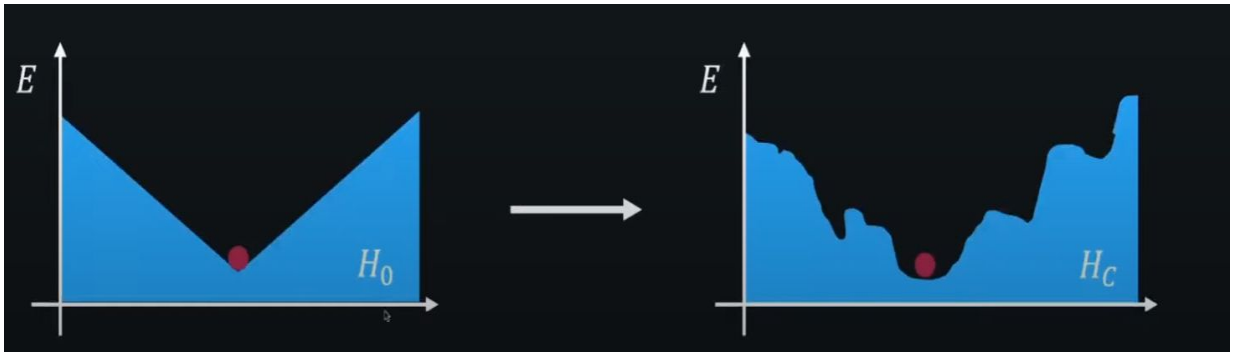
Hamiltonian operator

$$H_C = \sum_{i,j=1}^{n} \frac{1}{4} Q_{ij} Z_i Z_j - \sum_{i=1}^{n} \frac{1}{2} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) Z_i + \left(\sum_{i,j=1}^{n} \frac{Q_{ij}}{4} + \sum_{i=1}^{n} \frac{c_i}{2} \right)$$

Adiabatic Process and Trotterization

- The QAOA architecture is said to be a layerized variational form based on trotterized adiabatic process.
- We know from the schrodinger's equation that the evolution of a quantum system with a time independent Hamiltonian H = $|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle$
- The adiabatic theorem states that if the hamiltonian of a system in its ground state is perturbed slowly enough, the system remains in its ground state.
- Form of quantum computing that uses adiabatic theorem:
  - Encode problem as hamiltonian whose ground state is the problem solution.
  - Prepare quantum system in ground state of a simple hamiltonian.
  - Adiabatically evolve simple hamiltonian to problem hamiltonian.

Adiabatic Process and Trotterization Contd.

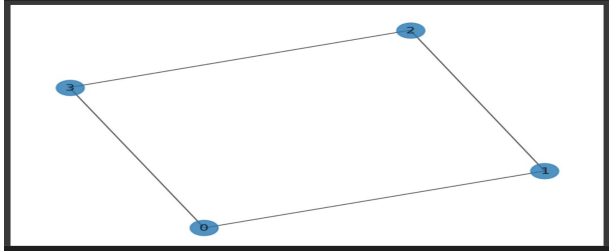


Trotter Suzuki Formula:
Approximates the exponential of a sum of matrices

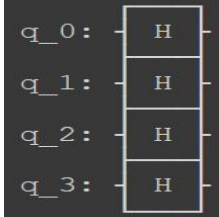
$$e^{-i(H_1+H_2)t} \approx \left(e^{-iH_1t/r}e^{-iH_2t/r}\right)^r$$

QAOA Implementation

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_nodes_from([0, 1, 2, 3])
G.add_edges_from([(0, 1), (1, 2), (2, 3), (3, 0)])
nx.draw(G, with_labels=True, alpha=0.8, node_size=500)
```

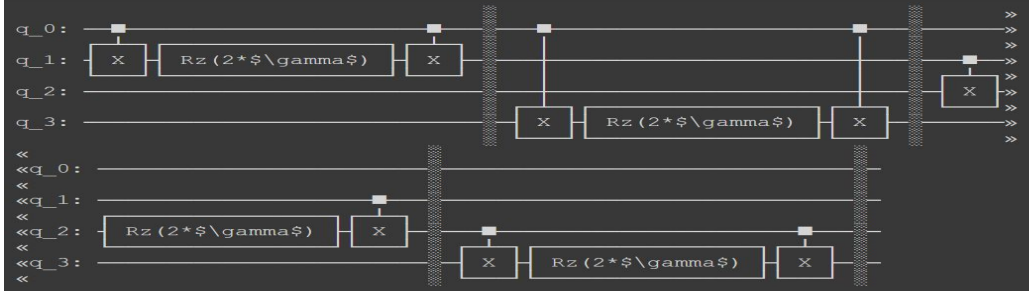



```
qc_0 = QuantumCircuit(nqubits)
for i in range(0, nqubits):
    qc_0.h(i)

qc_0.draw()
```



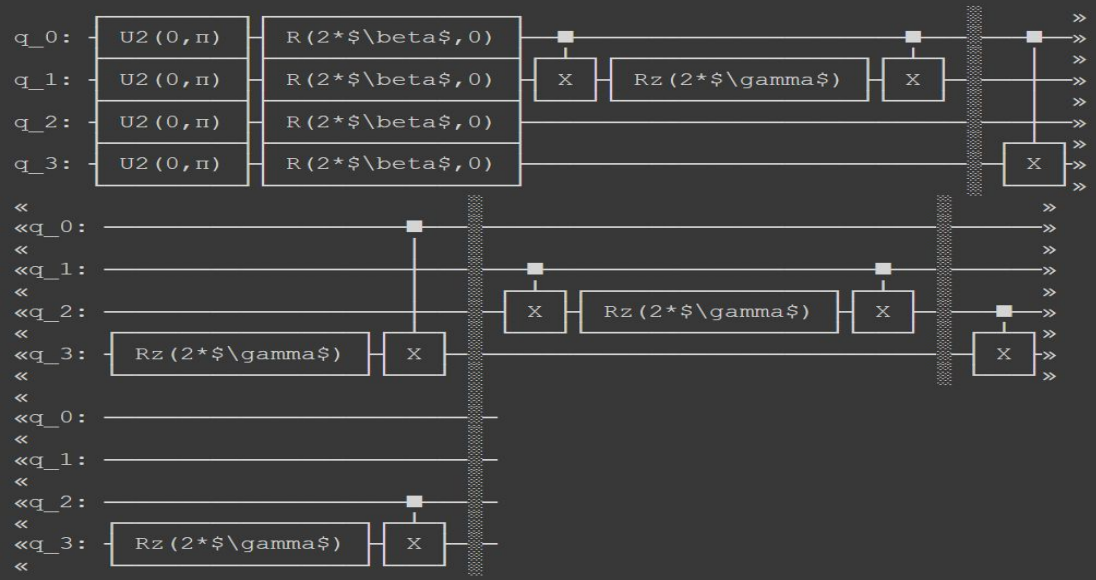

```
adjacency = nx.adjacency_matrix(G).todense()

nqubits = 4

beta = Parameter("$\\beta$")
qc_mix = QuantumCircuit(nqubits)
for i in range(0, nqubits):
    qc_mix.rx(2 * beta, i)

qc_mix.draw()
```


```
gamma = Parameter("$\\gamma$")
qc_p = QuantumCircuit(nqubits)
for pair in list(G.edges()):  # pairs of nodes
    qc_p.rzz(2 * gamma, pair[0], pair[1])
    qc_p.barrier()

qc_p.decompose().draw()
```



QAOA Circuit

```
qc_qaoa = QuantumCircuit(nqubits)

qc_qaoa.append(qc_0, [i for i in range(0, nqubits)])
qc_qaoa.append(qc_mix, [i for i in range(0, nqubits)])
qc_qaoa.append(qc_p, [i for i in range(0, nqubits)])

qc_qaoa.decompose().decompose().draw()
```

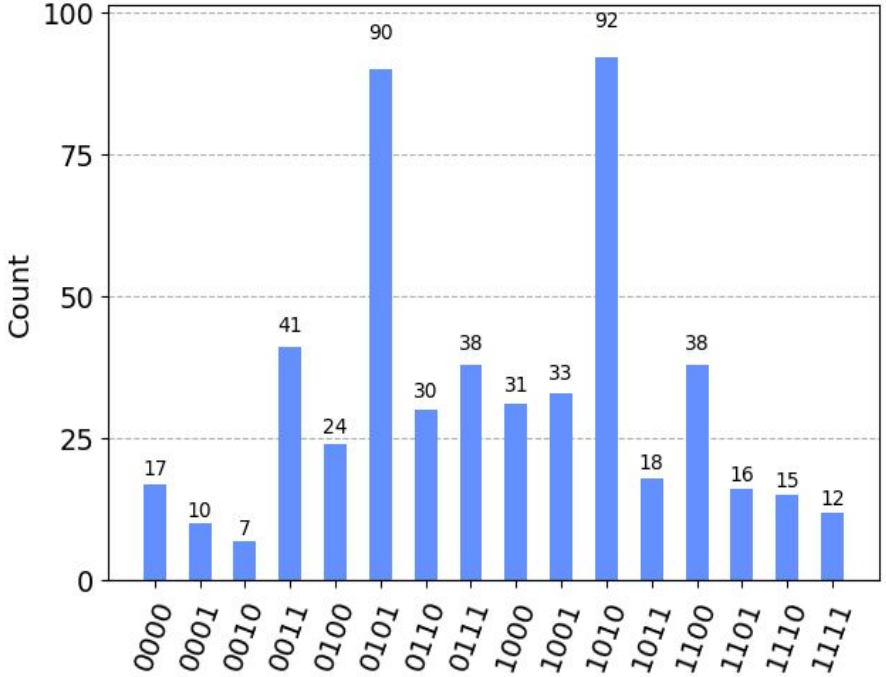


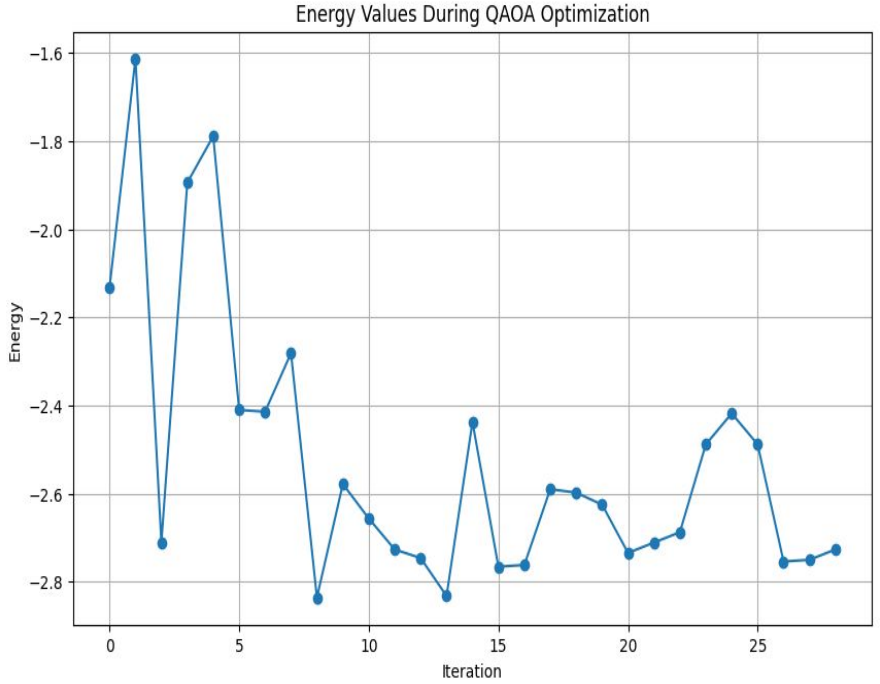
```python
from qiskit import QuantumCircuit, transpile
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor


def maxcut_obj(x, G):
    obj = 0
    for i, j in G.edges():
        if x[i] != x[j]:
            obj -= 1

    return obj

def compute_expectation(counts, G):
    avg = 0
    sum_count = 0
    for bitstring, count in counts.items():

        obj = maxcut_obj(bitstring, G)
        avg += obj * count
        sum_count += count

    return avg/sum_count

def create_qaoa_circ(G, theta):
    nqubits = len(G.nodes())
    p = len(theta)//2  # number of alternating unitaries
    qc = QuantumCircuit(nqubits)

    beta = theta[:p]
    gamma = theta[p:]

    # initial_state
    for i in range(0, nqubits):
        qc.h(i)

    for irep in range(0, p):
        # problem unitary
        for pair in list(G.edges()):
            qc.rzz(2 * gamma[irep], pair[0], pair[1])
        # mixer unitary
        for i in range(0, nqubits):
            qc.rx(2 * beta[irep], i)
    qc.measure_all()
    return qc

def get_expectation(G, p, shots=512):
    #backend = provider.get_backend('ibmq_qasm_simulator')
    backend = provider.get_backend('ibm_osaka')

    def execute_circ(theta):
        qc = create_qaoa_circ(G, theta)
        transpiled_qc = transpile(qc, backend)
        job = backend.run(transpiled_qc, shots=shots)
        job_monitor(job)  # Optional, to monitor job status
        result = job.result()
        counts = result.get_counts(qc)

        return compute_expectation(counts, G)
```

```python
from scipy.optimize import minimize


expectation = get_expectation(G, p=1)

res = minimize(expectation,
                           [1.0, 1.0],
                           method='COBYLA')
res
```

```
Job Status: job has successfully run
 message: Optimization terminated successfully.
 success: True
  status: 1
     fun: -2.5703125
       x: [ 1.977e+00  1.200e+00]
    nfev: 33
   maxcv: 0.0
```

Results


Count
100
75
50
25
0
17
10
7
41
24
90
30
38
31
33
92
18
38
16
15
12
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111


Energy Values During QAOA Optimization
Energy
−1.6
−1.8
−2.0
−2.2
−2.4
−2.6
−2.8
0
5
10
15
20
25
Iteration

Conclusion

- As this algorithm follows a adiabatic schedule, there are no guarantees in finite limits but the only guarantee is that as the depth of the circuit increases, we get a better approximation.
- This circuit is implemented on the ‘ibm_osaka’ device which has 127 qubits for 512 shots which reduced the effect of noise.
- This algorithm is sensitive to initialization but shows a lot of scope in solving QUBOs with the right ansatz and the right depth.