

Vanilla QAOA:

Components:

- QAA inspired: Discretized (Trotterized) version of Quantum Adiabatic Approximation
Repeated layers of $U=U_c U_m$, similar to the following equation for QAA (except f and g). (r is the number of layers)

$$\hat{U}(t) \approx \prod_{k=0}^{r-1} \exp \left[-i \hat{H}(k \Delta \tau) \Delta \tau \right] = \prod_{k=0}^{r-1} \exp \left[-i f(k \Delta \tau) \hat{H}_C \Delta \tau \right] \exp \left[-i g(k \Delta \tau) \hat{H}_M \Delta \tau \right] \quad (9)$$

- VQE method: Minimum expectation value of cost Hamiltonian is found (Variational Quantum Eigensolver feature i.e., expectation value \geq lowest energy value).
- Classical optimization: γ_k (counterpart of f) and β_k (counterpart of g) are tuned classically.

Note: the QAA feature in QAOA help us find a suitable initial trial function when we don't know any eigenstate of cost Hamiltonian (but we do know of mixer Hamiltonian)

Process:

1) Defining H_c and H_m

Define H_c according to the problem

Define a H_m that does not commute with H_c

Example H_c and H_m for MaxCut problem (i and j defining the ith and jth qubit):

$$\hat{H}_C = \frac{1}{2} \sum_{(i,j) \in \mathcal{E}} w_{ij} (I - Z_i Z_j),$$

$$\hat{H}_M = \sum_{j \in \mathcal{V}} X_j,$$

Where each binary variable $x_i = 0.5(I - Z_i)$ i.e., when state= (0 1), $x_i = 0.5(1 - (-1)) = 1$. And when state= (1 0), $x_i = (1 - 1) = 0$.

Making H_c the same as the objective function of MaxCut:

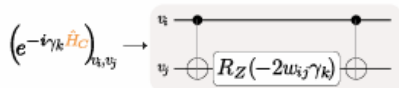
$$C(\mathbf{x}) = \sum_{i,j=1}^{|\mathcal{V}|} w_{ij} x_i (1 - x_j),$$

2) Creating the circuit ansatz containing U_c and U_m unitaries.

$$\hat{U}_C(\gamma) = e^{-i\gamma \hat{H}_C} = \prod_{i=1, j < i}^n R_{Z_i Z_j}(-2w_{ij}\gamma),$$

Cost interaction can be implemented using two CNOTs gates with one R_z gate (acting on the target qubit of CNOTs) in between.

Circuit ansatz U_c part:



$$\hat{U}_M(\beta) = e^{-i\beta \hat{H}_M} = \prod_{i=1}^n R_{X_i}(2\beta),$$

Mixer interaction can be implemented using rotation gate R_x .

Circuit ansatz U_m part:

$$(e^{-i\beta_k \hat{H}_M})_{v_i} \rightarrow v_i - R_X(2\beta_k) -$$

3) Defining the initial state

The initial state is *typically* defined as the highest energy state of the mixer for problems like MaxCut i.e., tensor products of $|+\rangle$ (highest energy state of the Pauli-X basis).

$$|s\rangle = |+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle,$$

$\sqrt{2^n}$ denominator is used to normalize the state i.e. to make the total probability 1. \mathbf{x} is the binary string of each combination of the states.

(Note: Here, the initial state being a highest energy eigenstate of the mixer (depending on sign convention) instead of the ground state still satisfies the adiabatic theorem, as simply an eigenstate of mixer is required as initial state- in QAOA it is not our objective to evolve into the corresponding eigenstate of the cost Hamiltonian. Rather, we utilize the variational principle: *the lowest energy state expectation value \geq ground state energy*)

4) Layers

- Total number of layers, p , should be at least 1.
- For each layer, define the variational parameters (p no. of parameters for U_C and p no. of parameters for U_M . Total = $2p$) γ and β , such that $\gamma_k \in [0, 2\pi)$ and $\beta_k \in [0, \pi)$, where k is the k th layer.
(Note: Due to the symmetry in the full QAOA circuit and the way the optimization landscape repeats, choosing $\beta_k \in [0, \pi)$ is often enough to cover all unique cases. This keeps the search space smaller.)

- The ansatz state after it goes through the repeated unitaries:

$$|\psi_p(\gamma, \beta)\rangle = e^{-i\beta_p \hat{H}_M} e^{-i\gamma_p \hat{H}_C} \dots e^{-i\beta_1 \hat{H}_M} e^{-i\gamma_1 \hat{H}_C} |s\rangle$$

5) Repeated measurements of the ansatz state

The final state is measured and the expectation value of H_C w.r.t. the ansatz state is calculated.

$$F_p(\gamma, \beta) = \langle \psi_p(\gamma, \beta) | \hat{H}_C | \psi_p(\gamma, \beta) \rangle$$

6) Classical optimization

The variational parameters are updated iteratively using a classical optimizer for the expectation value of the cost Hamiltonian w.r.t. the ansatz state to reach maximum.

$$(\gamma^*, \beta^*) = \arg \max_{\gamma, \beta} F_p(\gamma, \beta)$$

An analog version of QAOA was recently proposed.

QISKIT Simulation of QAOA MaxCut

Two different QAOA classes encountered:

| Source: qiskit.algorithms | Source: OpenQuantumComputing/QAOA |
|--|---|
| <p><i>qaoa = QAOA(sampler,optimizer, reps, shots)</i> paired with <i>qaoa.compute_minimum_eigenvalue(operator)</i></p> <ul style="list-style-type: none">• Initial state not defined (default?)• Operator defined as the problem (cost) Hamiltonian in Pauli Operator form• Mixer not defined (default?)• Sampler defined• Optimizer defined (default COBYLA())• reps defined (default 1)• shots optional• cvar not defined (default?)• backend not defined (default?)• noisemodel optional | <p><i>qaoa = QAOA(initialstate,problem,mixer,cvar)</i></p> <ul style="list-style-type: none">• Initial state defined• Problem (cost) Hamiltonian is in circuit form• Mixer defined• Sampler not defined (default?)• Optimizer optional (default COBYLA())• reps defined as depth in <i>qaoa.optimize(depth=maxdepth)</i><ul style="list-style-type: none">- Direct reps=maxdepth computation is not done because it becomes intractable for reps>1. Depth is increased iteratively.For reps=1, <i>qaoa.sample_cost_landscape()</i><ul style="list-style-type: none">- <i>qaoa.optimize(depth=maxdepth)</i> calls this function before iterative increasing• shots optional• cvar optional (default 1)• backend optional• noisemodel optional |

Performance

MaxCut:

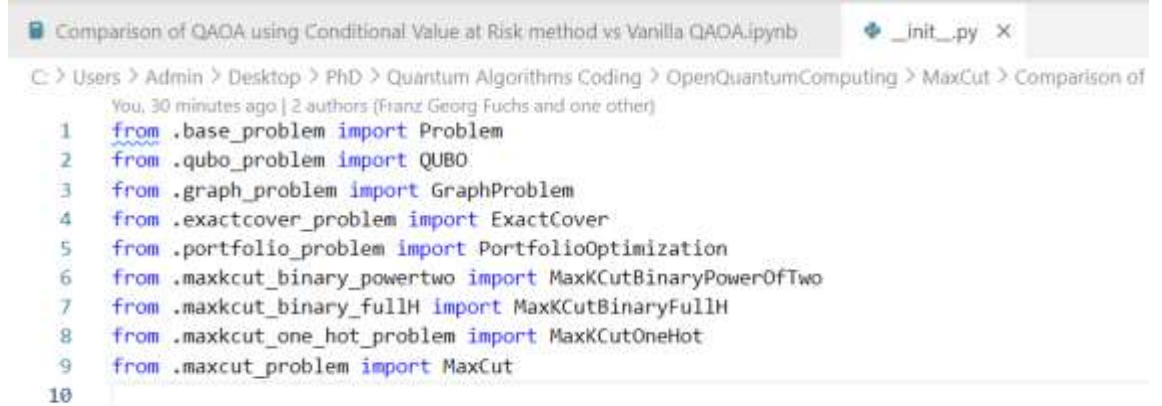
- **Comparison CVaR vs vanilla:** CVaR resulted in more accurate minimum cost than vanilla (esp. for lower depths). As p approached 10, approximation ratio for CVaR reached 1 whereas approximation ratio for vanilla 0.925.
- **Comparison Optimizers:**

Problems faced

1. In the MaxCut Problem Comparison of QAOA using CVaR and Vanilla QAOA code:

The MaxCut class was removed from the original repo OpenQuantumComputing/QAOA.

Solution: Forked the repo, cloned the forked repo and re-added the removed maxcut_problem.py file in the problems folder. From maxcut_problem.py imported MaxCut in the __init__.py file in the problems folder (line 9).



```
Comparison of QAOA using Conditional Value at Risk method vs Vanilla QAOA.ipynb  _init_.py X
C: > Users > Admin > Desktop > PhD > Quantum Algorithms Coding > OpenQuantumComputing > MaxCut > Comparison of
You, 30 minutes ago | 2 authors (Franz Georg Fuchs and one other)
1 from .base_problem import Problem
2 from .qubo_problem import QUBO
3 from .graph_problem import GraphProblem
4 from .exactcover_problem import ExactCover
5 from .portfolio_problem import PortfolioOptimization
6 from .maxkcut_binary_powertwo import MaxKCutBinaryPowerOfTwo
7 from .maxkcut_binary_fullH import MaxKCutBinaryFullH
8 from .maxkcut_one_hot_problem import MaxKCutOneHot
9 from .maxcut_problem import MaxCut
10
```

Some PennyLane QAOA Simulation Characteristics

PennyLane QAOA module offers built-in cost and recommended mixer Hamiltonians for different problems. We can also customize them, for example, by adding constraints or instantiating a different mixer Hamiltonian.

Adding a constraint: A reward or penalty for constraint can be added to the problem cost Hamiltonian. PennyLane allows arithmetic operations of Hamiltonian objects.

Example: For minimum vertex cover problem, **edge_driver()** cost Hamiltonian can be used to add constraint to the old cost Hamiltonian. Let's say from a multiple solution case, a solution can be favored by assigned it lower energy levels. In the following code, we are rewarding if vertices 0 and 2 are in the cover, i.e. 1010 for a 4 vertices graph.

Code for new cost Hamiltonian:

```
reward_for_meeting_constraint_h = qaoa.edge_driver(nx.Graph([(0,2)]), ["11"]) # when vertices/qubits 0,2 are
"11", the edge is assigned a lower energy than in case of other combinations of values (i.e. "00", "01", "10")

new_cost_h = cost_h + 2* reward_for_meeting_constraint_h
```

Miscellaneous

The nodes and edges of the graph are assigned in the gml file.

```
G=nx.read_gml("QAOA/examples/MaxCut/data/w_ba_n21_k4_0.gml")
```

```
graph [
  node [
    id 0
    label "0"
  ]
  node [
    id 1
    label "1"
  ]
  .
  .
  .
  edge [
    source 0
    target 7
    weight 0.6719596645247027
  ]
  edge [
    source 1
    target 4
    weight 0.5033779645445525
  ]
]
```