An Introduction to Quantum Approximate Optimization Algorithm using Qiskit

Sushant Ranjan
Department of Electrical and Computer Engineering, Fall 2022
Duke Graduate School
email address: sushant.ranjan@duke.edu

***Abstract*—Quantum Approximate Optimization Algorithm or QAOA can be used to solve combinatorial optimization problems. It is a hybrid algorithm that uses quantum and classical operations to solve the problem. This report provides an introduction to QAOA and shows how to use it to solve the Max-Cut problem using the IBM Quantum Experience Qiskit simulator. We will write our own functions for each step to understand the process. In the end, we will also use Qiskit's inbuilt packages to solve the same problem.**
All code used in this report can be found on GitHub: https://www.github.com/RanjanSushant/ece520_final_project

***Index Terms*—QAOA, Max-Cut, graphs, cost function, Qiskit, optimization**

I. Introduction

One metric to measure the performance of a quantum computer is "quantum advantage." It describes how better it performs than a classical computer for the same problem. Quantum computer scientists have devised various quantum algorithms which try to solve real-world problems with a noticeable speed up. One such algorithm is Quantum Approximate Optimization Algorithm or QAOA. QAOA belongs to a class of algorithms that solve combinatorial optimization problems. Such problems are really significant in situations such as supply chain, logistics, etc. Not only that, QAOA is one the algorithms that help us show quantum advantage. Combinatorial optimizations are NP-hard problems. This means that as the problem size grows, the time to reach a solution grows exponentially as well.

II. About QAOA

A. Analogy

Suppose you are planning a trip with your friends. You will want to be able to visit as many places as possible. However, some constraints (time, money, etc.) will be involved in making that decision. Making an itinerary such that the end goal is met while keeping the constraints in place is a type of optimization problem. Optimization plays a significant role in the industry and is applicable to many sectors. There are classical algorithms that solve the optimization problem, but as the size of the problem grows, it becomes harder to solve in polynomial time. This is where quantum computers come in. We can use quantum algorithms to get to our solution faster and solve larger combinatorial optimization problems.

B. Algorithm

QAOA is a hybrid variational algorithm . A variational quantum algorithm is a parameterized circuit whose parameters are modified after every iteration to find the optimal solution for that circuit. A noted example is Grover's algorithm.

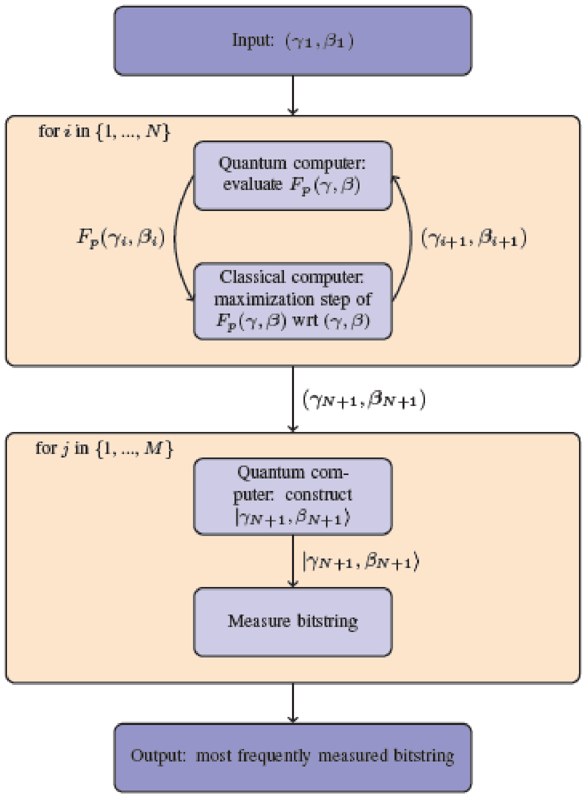

Input: (γ_1, β_1)
for i in $\{1, ..., N\}$
Quantum computer: evaluate $F_p(\gamma, \beta)$
$F_p(\gamma_i, \beta_i)$
$(\gamma_{i+1}, \beta_{i+1})$
Classical computer: maximization step of $F_p(\gamma, \beta)$ wrt (γ, β)
$(\gamma_{N+1}, \beta_{N+1})$
for j in $\{1, ..., M\}$
Quantum computer: construct $|\gamma_{N+1}, \beta_{N+1}\rangle$
$|\gamma_{N+1}, \beta_{N+1}\rangle$
Measure bitstring
Output: most frequently measured bitstring

Fig. 1. Flowchart of the QAOA algorithm. The upper box outlines the hybrid classical-quantum approach, where the quantum computer evaluates the objective function and the classical computer optimizes it in a step-wise manner for a fixed number of steps N. Given the optimized parameters, the quantum computer then constructs the corresponding state and measures it, returning a bitstring. This procedure is repeated a fixed number of steps M and the most frequently measured bitstring is the output of the algorithm.

QAOA depends on two parameters, γ and β, which are optimized after every iteration of the circuit.Since QAOA is hybrid it also utilizes classical computing. Optimizing and updating the parameters is where the classical operations come in. The quantum algorithm runs for an initial state or an ansatz state. Then, based on the outputs, the classical optimizer generates new values of the two parameters to be used for the next iteration. As the number of iterations increases our approximation gets better and better.

C. *Circuit*

A QAOA circuit consists of four parts. These four parts contribute to different parts of the algorithm and make use of one-bit gates, two-qubit gates, and classical optimizer.

1) Hadamards: To put all the qubits into superposition.

2) Cost circuit: The cost circuit or the objective circuit is the mapping of the problem into an Hamiltonian which in turn is made into a Quantum Circuit.

3) Mixer circuit: As the name suggests, it keeps things mixed. This makes sure that our probability distribution is affected even if the input is an eigenstate of the circuit so that we avoid wasting iterations.

4) Classical Part: The circuit parameters are optimized using a classical optimizer such as COBYLA, which is what I have used for this project. COBYLA stands for Constrained Optimization By Linear Approximation and is a classical optimizer available from the scipy package in python.

III. Application

QAOA can be applied to a variety of existing optmization problems such as Max-Cut, Travelling salesman, etc. The general process of constructing a Hamiltonian for an abitrary function involves mapping boolean and real functions to diagonal Hamiltonians acting on qubits. We start by performing the Fourier analysis of the desired function and map the resulting expansion on to a hamiltonian made up of Z gates.

Perform Fourier expansion of the desired function:

$$f(x) = \sum_{S\subseteq[n]} \widehat{f}(S)x^S$$

The unique n-qubit Hamiltonian representing f is given by:

$$C = \sum_{S\subseteq[n]} \widehat{f}(S)\prod_{j\in S} Z_j$$

Fig. 2. General process of mapping functions to Hamiltonians

IV. Max-Cut Problem

The Max-Cut problem involves dividing any given graph into disjoint sets of nodes by cutting the minimum number of edges possible. There is also a weighted version of this problem where are all edges have some weight associated with them. The Max-Cut objective or cost function is classically given by

1) Constructing the Hamiltonian: The Hamiltonian for the Max-Cut problem comprises of Pauli Z gates.
The Pauli Z operator gives a negative when the input qubit is 1 and positive when it is 0.The equation can even be generalized as we can see next

$$\max_{\mathbf{s}} \frac{1}{2}\sum_{ij\in E}(1 - s_i s_j) \qquad s_i \in \{-1, +1\}$$

Fig. 3. Max-Cut objective

Note that it has eigenvalues -1, +1 with eigenvectors being computational basis states

$$Z|0\rangle = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

$$Z|1\rangle = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\-1\end{bmatrix} = (-1)|1\rangle$$

Therefore,

$$Z|x\rangle = (-1)^x|x\rangle \quad x \in \{0,1\}$$

Fig. 4. Pauli Z operator

The final Max-Cut Hamiltonian can be constructed by mapping the si variables to the Z operator eigenvalues. This will result in the following

$$C = \frac{1}{2}\sum_{ij\in E}(I - Z_i Z_j)$$

Fig. 5. Max-Cut Hamiltonian

We will use this Hamiltonian to create our cost circuit when simulating this problem in Qiskit.

The mixer circuit of Max-Cut is Pauli X gates applied to all the qubits.

$$B = \sum_i X_i$$

Fig. 6. Max-Cut Mixer

V. Implementation

The next two sections cover the implementation of the Max-Cut problem using QAOA. We implemented the Max-Cut using Jupyter notebooks on https://lab.quantum-computing.ibm.com/ in two ways. First we implemented the cost Hamiltonian we calculated above using Qiskit. Then we implemented it using built-in library functions from Qiskit. The code for both of them is stored in the link mentioned in the abstract.

VI. From scratch

In order to implement our Max-Cut Hamiltonian from scratch we first need to create the graph on which we want to perform the Max-Cut. However, it is good practice to first import all the relevant packages before we start doing anything else. Next, we create the graph.

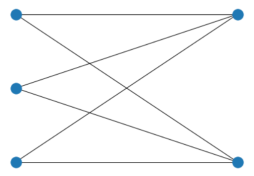


Fig. 7. Problem graph with 5 nodes and 6 edges

For the sake of simplicity we have laid out the graph in a bipartite fashion where we can see the obvious result. For the next step, we create the cost circuit by appending CNOTs and RZs to all the edges. The RZ is a rotating Z gate. For our circuit the angle of rotation depends on the parameter γ. The overall operator is given by the equation

$$e^{-i\gamma C} = e^{-i\gamma \frac{1}{2} \sum_{ij \in E}(I - Z_i Z_j)}$$

Fig. 8. Cost operator from Hamiltonian

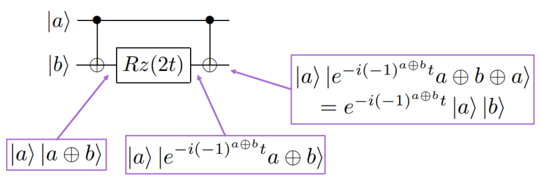

$|a\rangle$
$|b\rangle$
$Rz(2t)$
$|a\rangle\, |e^{-i(-1)^{a\oplus b}t} a \oplus b \oplus a\rangle$
$= e^{-i(-1)^{a\oplus b}t}\, |a\rangle\, |b\rangle$
$|a\rangle\, |a \oplus b\rangle$
$|a\rangle\, |e^{-i(-1)^{a\oplus b}t} a \oplus b\rangle$

Fig. 9. CNOT and RZ for every edge. Similar gates will be applied to all edges

Applying the above to all the edges of our graph and using γ=π/3 we get

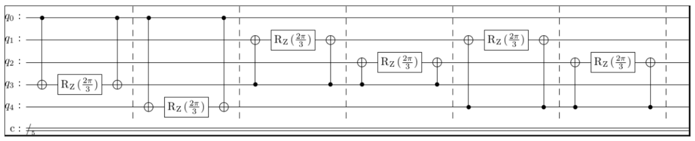


Fig. 10. Cost Circuit

Now that we are done with our cost circuit, we will build our mixer circuit which is given by the expression

$$e^{-i\beta B} = e^{-i\beta \sum_j X_j}$$

Fig. 11. Mixer Circuit

Applying this we get

Using all of the above we can now build our QAOA circuit by concatenating them. However, before we need to put all our qubits into superposition so we apply Hadamard to all of them. Then, we first apply the cost circuit followed by the

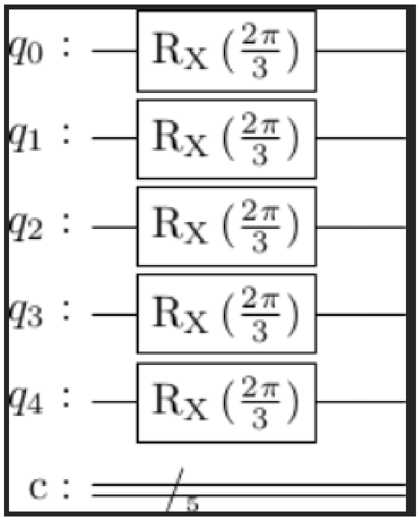


Fig. 12. Mixer Circuit

mixer circuit, The resulting final QAOA circuit for Max-Cut on our graph will look like

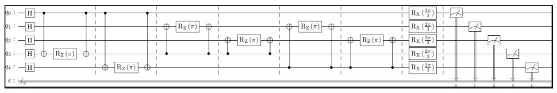


Fig. 13. QAOA Circuit for the graph we created

We run circuit on Qiskit's QASM_simulator to get our results. We run multiple iterations of the circuit. This increases our chances of better approximation. The entire parametrized circuit is passed as a blackbox to COBYLA, which is the classsical optimizer we have used to get optimal values for γ and β. To begin minimizing, COBYLA needs an initializing point. Depending on how we choose the initial state will affect the efficiency and performance of our algorithm. A good initial state will help us reach the solution faster. Choosing an initial state is a separate in and of itself. Our project uses a initial state calculated using the state vector simulator of Qiskit but we will not go into the details here. Running the optimizer with COBYLA will give us the optimal values for our parameters which we cna plug in to the circuit to find the solution to our Max-Cut problem for the graph. The output will be in the format of bitstrings and counts which can plot. The result will look similar to

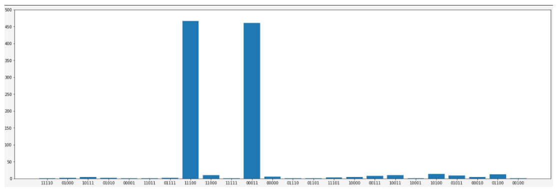


Fig. 14. Resulting plot for bitstrings and counts

As we can see the bitstrings 11100 and 00011 get the most number of counts which is to be expected because the graph gets divided best by cutting nodes 0,1, and 2 in part1 and 3, and 4 in part2. These two bitstrings denote exactly that and hence are the best solution. We can also plot the energies to see what number of cuts gave us the best solution

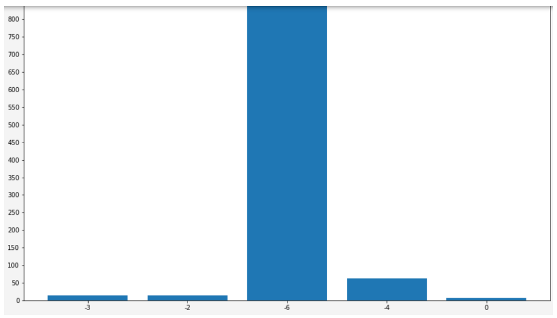


Fig. 15. Number of cuts

VII. Qiskit Library

The second of implementing Max-Cut is to use Qiskit's built-in library functions. We will be using the same graph so that we can compare outputs but we will add the option of weights on edges to be able to expand on it in future works. For simplicity and purposes of this project we will keep the weight fixed to 1. We create the graph and color the nodes red to show they are not disjoint yet.

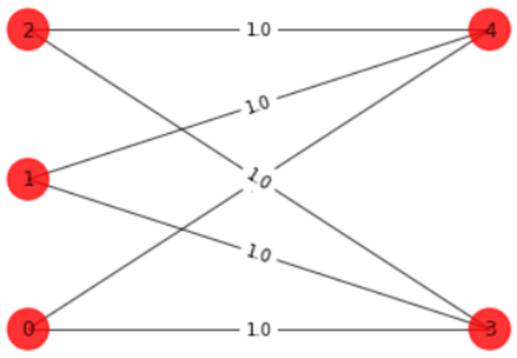
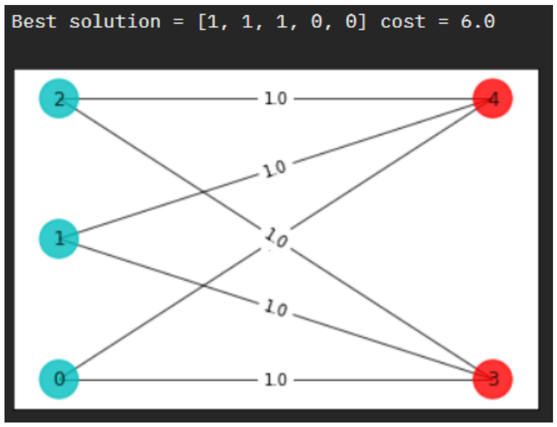


Fig. 16. Weighted graph

This graph is effectively same as the graph in the previous experiment. We have just added a weight to each edge but they don't really affect our calculations as all have same weight.

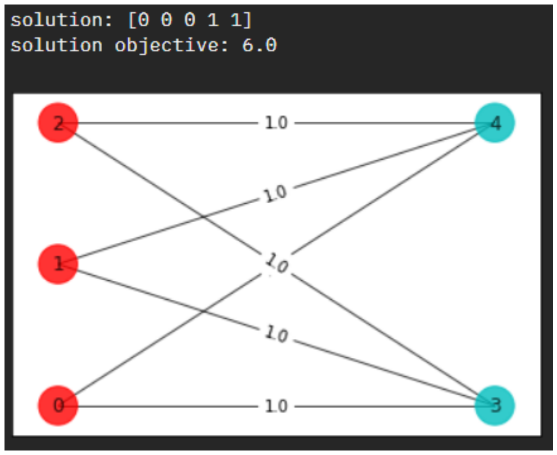
Next we calculate the solution using brute force. Since this is a small graph, it will be quick. This output will be what we expect from the QAOA as well. Note that colors are meant to depict the difference between two sets of nodes and can be the other way around. Same is applicable to bitstrings as well. 1 and 0 denote different partition.



```
Best solution = [1, 1, 1, 0, 0] cost = 6.0
```

Fig. 17. Expected solution

To create the Hamiltonian we first have to take the output of the MaxCut library function and convert it to a quadratic program. Then we take this output and create the ising Hamiltonian using it. If we print it we see it looks quite similar to our cost circuit we made from scratch. Next we use the NumPyMinimumEigenSolver and the MinimumEigenOptimizer to get the lowest eigenstate. Executing all the functions and printing out the solutions we get



```
solution: [0 0 0 1 1]
solution objective: 6.0
```

Fig. 18. Final result

We note that the output here is similar to what we expected i.e. what we got from the brute force method.

Comparing the results from the simulation we did before the one now we can see that they are quite similar. QAOA can be used to solve MaxCut on a given graph. Implementing from scratch helps us understand how the algorithm works and when we eventually use the library functions we can utilize them the best

VIII. Discussion and Future work

As we can see from the experiments we performed that QAOA is quite a interesting algorithm. Not only to prove quantum advantage but solving real world optimization problems as well. However, there are still a few caveats that need work. Currently, the algorithm depends significantly on the classical optimizer being used to update γ and $beta$. That limits the performance of our algorithm. Also, deciding on a good ansatz state is crucial to getting correct solutions at higher speeds. Future work revolves around improving upon these while using QAOA to solve other classical optimization problems.

References

[1] https://www.youtube.com/@RuslanShaydulin

[2] The Qiskit Textbook: $https : //learn.qiskit.org/course/ch - applications/solving - combinatorial - optimization - problems - using - qaoa$

[3] Farhi and Goldstone, https://arxiv.org/abs/1411.4028

[4] Tse, Mountney, Klein, and Severini, https://arxiv.org/abs/1812.03050