Implementation of Quantum Approximate Optimization Algorithm

Maisha Binte Rashid

August 10, 2022

1 Project Description

1.1 Overview

In quantum computing, one way of solving combinatorial optimization problems is to use Quantum Approximate Optimization Algorithm (QAOA). With the development of quantum computing in recent years, QAOA has become a robust algorithm. This approach establishes connections between essential elements like the quantum circuit and the Hamiltonian. In computer science, there exist different optimizing algorithms. Using QAOA, we can find optimal solutions for problems like finding optimal bit strings and minimum vertex cover. This, therefore, served as the motivation behind this project. Furthermore, we can identify more effective approaches to solve quantum problems by examining this algorithm and altering the parameters.

In this project, I used QAOA to find minimum vertex cover. Minimum vertex cover means a group of vertices where each graph edge includes at least one of the vertices in the cover; thus, every edge of the graph is covered by the vertices. Solving graph problems has become an essential topic in quantum computing. To find the minimum vertex, I constructed a quantum circuit. We can think of quantum circuits as Hamiltonian. Here, I used cost and mixer Hamiltonian to build a quantum circuit. Cost Hamiltonian refers to the cost of each state, and mixer Hamiltonian is the sum of Pauli-X operations of each vertex of the graph. To build layers in the circuit, I used the cost layer and mixer layer, which take cost Hamiltonian and mixer Hamiltonian as inputs. After that, I used GradientDescentOptimizer to find the optimal parameter. Lastly, I found the vertices for minimum vertex cover by optimizing the circuit using the optimal parameters. I used the article 'Intro to QAOA' as my motivation for this project [1].

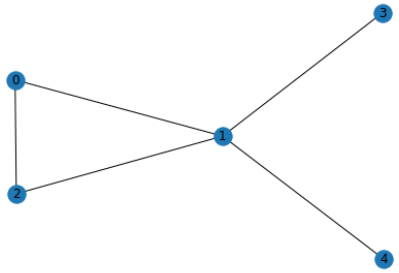

Figure 1: The undirected graph

1.2 Description

To implement QAOA, I used Pennylane's built-in QAOA functionality in this project. I had to install Pennylane version - 0.24.0 in my Google Colab notebook. I needed Pennylane's built-in QAOA function, so I imported it from Pennylane. I imported pyplot from matplotlib to plot the necessary charts for this project. I imported networkx to draw the graph from which I would find minimum vertex. Figure 1 shows the graph I used to find the minimum vertex cover. To implement QAOA, I need to create a quantum circuit. To begin with, after generating the graph, I defined cost Hamiltonian and mixer Hamiltonian. To create cost and mixer Hamiltonian I used Pennylane's built-in function 'min_vertex_cover'. For cost Hamiltonian it returned the ground state of all the vertices.

Another fundamental element of a quantum circuit is - layers. Layers are used to perform repeated applications in quantum algorithms. Using the functions cost_layer() and mixer_layer(), the QAOA module can construct the cost and mixer layers directly. Cost and mixer layer take cost Hamiltonian and mixer Hamiltonian as input states. After that, to create a variational circuit, I needed to define the number of wires. In quantum computing, wires refer to qubits which are the basic unit of information [2]. I initialized a number of wires = 5, the same as the number of vertices. While creating the circuit, there's a Hadamard gate installed in each wire. The layer function passes parameters into each layer of the circuit. To run a quantum circuit, we need a device. For that, I used an instance of Device class. In this project, to run the circuit, I described, I used 'default. qubit' as a simulator. I tried to use 'qulacs.simulator,' but there were some plugin issues for which I could not use it. So, my task was to optimize the circuit parameters after creating the circuit. One way of optimizing the circuit parameters is optimizing the cost function. I used expval() function as cost function, which returns the expected value of cost Hamiltonian.

As I wanted to optimize the cost function, I used GradientDescentOptimizer(). This is a built-in function in Pennylane. For the optimizer, I used step size =

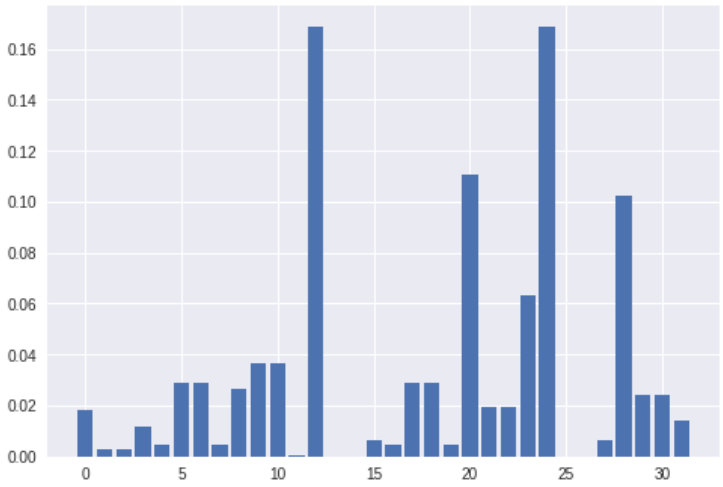

Figure 2: The bar graph of probability of each bit string using GradientDescentOptimizer where step size = 70

70 and initialized the parameters to 0.5. Choosing the correct value for initialization is quite difficult as there is a possibility of getting stuck in local minima. I ran the optimizer 70 times to find the optimal parameters. Using the optimal parameters, I ran the QAOA circuit again and measured the probability of each bit string. The bit strings with the highest probabilities are the vertices for vertex cover for the graph. In Figure 2, we can see that 12 and 24 have the highest probabilities. The 5-bit bitstring for 12 is 01100. It means that nodes 1 and 2 are the vertices that cover the graph. Again, the 5-bit bitstring for 24 is 11000. It means that nodes 0 and 1 are the vertices that cover the graph. From the bar graph, we can conclude that either nodes 1 and 2 or nodes 0 and 1 are the solutions for minimum vertex cover for the graph in Figure 1.

1.3 Experiments and Result

In the previous section, I used GradientDescentOptimizer for optimizing the cost function. I used another optimizer which is AdamOptimizer to evaluate the performance between two optimizers in terms of giving accurate result. I used 70 step size to ran both the optimizers. For GradientDescentOptimizer the optimal parameters I got are given below -
[[0.49111.108]
[0.570.966]]
The optimal parameters I found for AdamOptimizer are-
[[0.7270.748]
[0.4740.781]]
I obtained the exact probabilities for each bitstring for both optimizers using

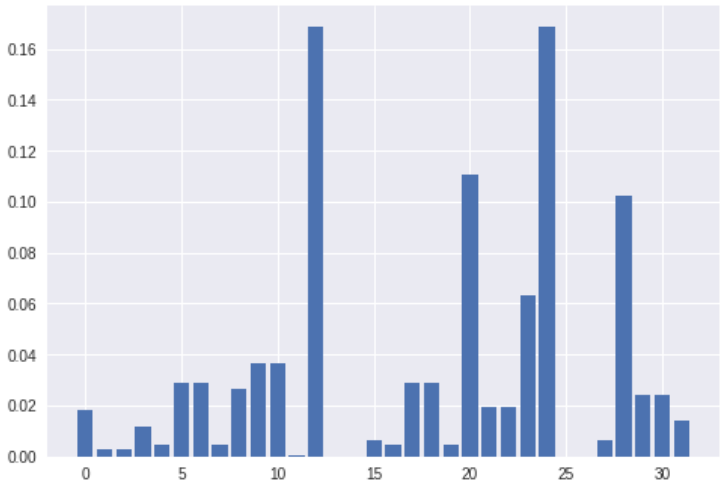

Figure 3: The bar graph of probability of each bit string using AdamOptimizer where step size = 70

these parameters, as shown in Figure 2 and Figure 3. But when I lessened the number of step size to 3, the result changed for AdamOptimizer, as shown in Figure 4. It shows that 28 has the highest probability, but in Figure 2 and 3, we see that 12 and 24 have the highest probability. This happened because when the number of step size was 30, the optimizer did not reach the global minima, and as a result, it did not find the optimal solution.

2 Future Work and Conclusion

QAOA is able to find minimum vertex cover from any given graph through optimization techniques. In the future, I want to create a dataset where there will be graphs and corresponding vertex covers. By using this dataset, my goal is to train the dataset with a model so that the system will be able to find minimum vertex covers for any graphs. Using QAOA we can solve other graph problems in quantum computing as well.

References

[1] `https://pennylane.ai/qml/demos/tutorial_qaoa_intro.html`

[2] `https://pennylane.readthedocs.io/en/stable/introduction/circuits.html`

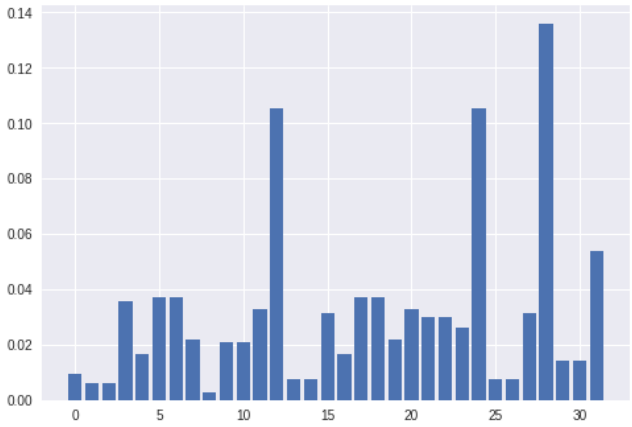

Figure 4: The bar graph of probability of each bit string using AdamOptimizer where step size = 30