QAOA for the Maximum Cut Problem

Girish Ganesan

May 3, 2021

Abstract

Quantum approximate optimization algorithms (QAOA) are a class of quantum algorithms applicable to combinatorial optimization problems. In this paper, I present a general overview of the process of adapting a problem to a QAOA-based approach. Then, I apply QAOA to the Max-Cut Problem for unweighted graphs, showing the process of developing the appropriate circuit in Qiskit and assessing the validity of my measurements.

1 Overview

Currently, the quantum computers in development are extremely limited in their capabilities. A number of engineering challenges have limited progress in building quantum computers suitable for running the most important and promising applications physicists have envisioned. The implementation of Shor's algorithm for factorization of large primes and efficient quantum chemistry simulations both rely on large numbers of qubits, each of which needs to be error-checked and kept in the strictest isolation possible for the duration of the computation.

In contrast, QAOA is an application that is low-cost in the number of qubits, can achieve state-of-the-art performance on par or exceeding that of classical computers, and is achieved through interfacing with classical computation, removing the need for excessive fault tolerance. We set up the combinatorial optimization scenario as follows. A discrete variable x can assume values in the domain S and is equipped with a cost function $C(x) : S \to \mathbb{R}$ that we wish to maximize. The QAOA approach is to develop an "ansatz", or a parametrized circuit architecture, informed by the nature of the problem. We then shift focus to find the optimal parameters such that when the state $|x\rangle$ is prepared as input, the circuit returns the state $|C(x)\rangle$ as measured output. This optimization process is iterative, with each run of the algorithm getting closer to the true result.

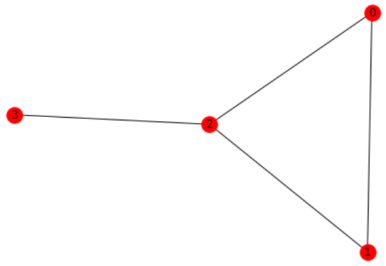
Literature Review

QAOA was originally propsed in the paper "A Quantum Approximate Optimization Algorithm" by Farhi et al (2014)[1]. To bolster the case in favor of using QAOA as a practical, robust approach to solve combinatorial optimization problems, the authors present two notable arguments. The first lies in their investigation of k-regular graphs. A k-regular graph is one in which every vertex has exactly k neighbors. By classifying all types of 3-regular graphs, they present a convincing argument that the approximation ratio of a QAOA Max-Cut circuit has a worst-case value of 0.6924. The approximation ratio for some possible partition of the graph (in the Max-Cut setting) is the value of the cut indicated by the partition divided by the optimal value of the objective, i.e. the maximal cut possible. Worst-case ratios are also provided for special kinds of n-regular graphs for $n > 3$ from a rigorous mathematical standpoint, adding credibility to the capacity of QAOA to solve NP-hard problems in average-case polynomial time.

Moreover, they analogize the QAOA parameter optimization process to adiabatic evolution. The quantum adiabatic algorithm involves finding a high-energy eigenstate of some Hamiltonian by simulating a different time-dependent Hamiltonian on a quantum computer[2]. This algorithm dates back to 2000 and was discovered by the same author; framing QAOA as a continuation (and in many ways, an improvement) of that work builds confidence that QAOA and other approximate optimization techniques may be the first sign of quantum supremacy in the "noisy intermediate-scale quantum" (NISQ) era.

2 QAOA for Max-Cut: Derivation

The max-cut problem for unweighted, undirected graphs can be stated as follows: color the nodes of graph G with two colors such that the number of edges connecting nodes of different colors is maximized. For the graph shown below, the partition $\{0, 1\}, \{2, 3\}$ yields a cut value of 2, while the partition $\{0, 3\}, \{1, 2\}$ yields a cut value of 3 which is maximal. The procedure of partitioning the node



set of the graph into two disjoint sets can be numerically representing as assigning $s_i \in \{-1, +1\}$ for each node in the graph. Then, summing over all edges

in the graph (and accounting for double counting the same edge), the cut value relating to any given partition of G is

$$\frac{1}{2}\sum_{(u,v)\in G}(1-s_u s_v)$$

.

Finding the Hamiltonian

To translate this problem into the language of the quantum computer, we need to envelop this information into a Hamiltonian matrix C, such that

$$C\left|x\right\rangle = C(x)\left|x\right\rangle$$

We first observe that the Pauli-Z matrix has eigenvalues $+1$ and -1 by inspection, with z-basis vectors as its eigenstates. Then, define $C := \frac{1}{2}\sum_{(u,v)\in G}(1-Z_u Z_v)$ where Z_k is a Pauli-Z gate operating on the k-th qubit of the input state. This matrix C has the desired property.

$$C\left|x\right\rangle = C\left|x_0 x_1 ... x_n\right\rangle \forall x_i \in \{0,1\}$$

$$= \frac{1}{2}\sum_{(u,v)\in G}(1-Z_u Z_v)\left|x_0 x_1 ... x_n\right\rangle$$

$$= \frac{1}{2}\sum_{(u,v)\in G}\left|x_0 x_1 ... x_n\right\rangle - \left|x_0 x_1 ...\right\rangle (Z_u)\left|x_u\right\rangle\left|x_{u+1} ... x_{v-1}\right\rangle (Z_v)\left|x_v\right\rangle\left|...x_n\right\rangle$$

$$= \frac{1}{2}\sum_{(u,v)\in G}\left|x_0 x_1 ... x_n\right\rangle - (-1)^{x_u}(-1)^{x_v}\left|x_0 x_1 ... x_n\right\rangle$$

$$= (\frac{1}{2}\sum_{(u,v)\in G}(1-(-1)^{x_u}(-1)^{x_v}))\left|x_0 x_1 ... x_n\right\rangle = C(x)\left|x\right\rangle$$

Note that in the final step, any term of the summation yields 0 if and only if $x_u = x_v$ and 1 if and only if $x_u \neq x_v$. Since this Hamiltonian represents the cost function, it is referred to as the cost Hamiltonian. We also define a mixer Hamiltonian M, which will serve as an additional layer in the final quantum circuit that separates each sequence of cost Hamiltonian gates, as the sum of Pauli-X gates on each qubit:

$$M = \sum_{i}^{n} X_i$$

Note that for a graph with $|G| = n$ vertices, each of these matrices is $2^n \times 2^n$ in dimensions. By describing them by summation, we avoid having to explicitly compute any elements of these matrices in the upcoming steps.

Preparing the Ansatz

With these two Hamiltonians in hand, we can now proceed to the construction of the ansatz circuit for the Max-Cut QAOA problem. An "ansatz" is a general structure defined for a circuit. Some of the gates will be parameterized, and over the course of the optimization process we will find the best parameters for this ansatz to obtain near-optimal results.

The QAOA setup requires us to apply alternating layers of unitary operators on the input state, which is an equal superposition. In other words, the state at the end of the circuit's run should be

$$= e^{-i\beta_p M} e^{-i\alpha_p C} ... e^{-i\beta_1 M} e^{-i\alpha_1 C} H^{\otimes n} |0\rangle$$

Here, the variable p simply represents the number of repeated layers of cost and mixer gates that the ansatz contains. This is determined in the selection of the ansatz at the very start and not changed during the run. Increasing p generally adds more power and expressivity to the circuit, at the cost of an increased runtime and a potentially prohibitive increase in cumulative error due to the increased number of fault-prone gates.

Let us first examine the expressions of the form $e^{-i\alpha M}$. Substituting in our explicit definition for M,

$$e^{-i\alpha M} = e^{-i\alpha \sum_{j=1}^{n} X_j}$$

$$= e^{-i\alpha X_1} e^{-i\alpha X_2} ... e^{-i\alpha X_n}$$

$$= \Pi_{j=1}^{n} e^{-i\alpha X_j}$$

Then, this part of the ansatz corresponds to a series of qubit-by-qubit rotations of angle 2α in the x-basis. A more complicated situation arose with operators of the form $e^{-i\beta C}$. Note first that for the purposes of maximization, the coefficients on the summation and constants inside are irrelevant. Then, we can express the objective $\frac{1}{2}\sum_{(u,v)\in G} 1 - Z_u Z_v$ more succinctly as $C = \sum_{(u,v)\in G} -Z_u Z_v$. It should be evident that maximizing the action of the first Hamiltonian on the equal superposition state maximizes the second as well. Then, we can explore how the operator $e^{-i\beta C}$ acts on the four two-qubit states x_u and x_v could possibly be in.

$$e^{-i\beta C} |00\rangle = e^{i\beta Z_u \otimes Z_v} |00\rangle = e^{i\beta} |00\rangle$$

$$e^{-i\beta C} |01\rangle = e^{i\beta Z_u \otimes Z_v} |01\rangle = e^{-i\beta} |01\rangle$$

$$e^{-i\beta C} |10\rangle = e^{i\beta Z_u \otimes Z_v} |10\rangle = e^{-i\beta} |10\rangle$$

$$e^{-i\beta C} |11\rangle = e^{i\beta Z_u \otimes Z_v} |11\rangle = e^{i\beta} |11\rangle$$

Then, the action of operator $e^{-i\beta C}$ on state $|x_u x_v\rangle$ for $x_u, x_v \in \{0, 1\}$ is

$$e^{i(-1)^{x_u \oplus x_v}\beta} |x_u x_v\rangle$$

The state remains the same albeit with a phase shift dependent on the two qubit values; if $x_u = x_v$, the phase shift is $e^{i\beta}$, else it is $e^{-i\beta}$. The sequence of gates that achieves this result is $CNOT(u \to v) \otimes R_z(2\beta) \otimes CNOT(u \to v)$, where the R_z gate represents a rotation by the specified angle in the z-basis.

3 QAOA for Max-Cut: Implementation

My full code is available here:
https://colab.research.google.com/drive/1Bn_XCxoIdZVLRiPU_MsHozQikNMNS_4k?usp=sharing
Outputs have been saved for easy viewing but the notebook is no longer linked to my IBM account.

The method GET_MAXCUT_ANSATZ returns a QuantumCircuit parameterized by the values passed into it. The value of p is inferred from the number of angle parameters provided.

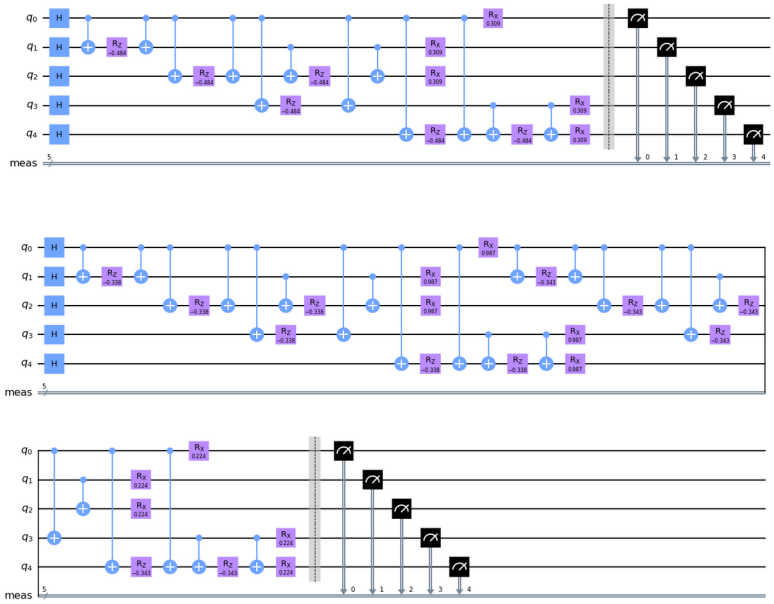
```
def get_maxcut_ansatz(graph, params):
    # Prepare the circuit ansatz.
    n_vertices = 1 + max([max(e) for e in graph.edges])
    qaoa = QuantumCircuit(n_vertices)
    p = len(params)//2
    alpha = params[:p]
    beta = params[p:]
    # (1) Place all qubits into equal superposition.
    for i in range(n_vertices):
        qaoa.h(i)
    # (2) Arrange layers of alternating rotations.
    for layer in range(p):
        for q1, q2 in graph.edges:
            qaoa.cx(q1, q2)
            qaoa.rz(alpha[layer], q2)
            qaoa.cx(q1, q2)
        for i in range(n_vertices):
            qaoa.rx(beta[layer], i)
    qaoa.measure_all()
    return qaoa
```

In the first step, a Hadamard transform is applied to the input qubits in the $|00...0\rangle$ state, placing them into an equal superposition. Next, p layers of the alternating CNOT-RZ-CNOT and RX gates are added. Finally, all qubits are measured and the observed values are stored in classical registers.

For a graph with 5 vertices and $p = 1$, the ansatz looks like so.

Note the cascade of CNOT-RZ-CNOT units, followed by a string of RX rotations on each wire. At the end, all qubits are measured. In contrast, consider this circuit for $p = 2$.

As expected, it has two layers of alternating cost and mixer gates. Note that the parameter values for the RZ gates differ across layers. The same applies for the RX gates.



The iterative workflow of the program was to first make measurements using the default trial parameter values assigned to the ansatz. Each set of n measurements was interpreted as a partition of the node set into two groups, and the cut value of this partition was calculated. Over a large number of shots, this process was repeated until all shots were exhausted, when the cost function returns the "expected cut value" produced by running the quantum circuit to find an adequate partition. This was all packaged into a function that represented a mapping of $2p$ parameters to a scalar value (expected cut value) and fed into the Scipy MINIMIZE utility.

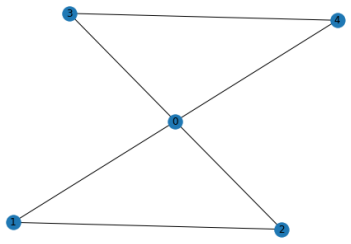
The OPTIMIZE_MAXCUT function is responsible for encapsulating this optimization routine. It uses the COBYLA algorithm, the default suggested by Scipy. Note that this method returns three pieces of information: the best partition found, the cut value of that partition, and the full frequency distribution of each outcome for further analysis and visualization.

Methods to create random graphs and calculate maximum cuts by brute force (iterating over all possible partitions) are also included for testing and debugging purposes.

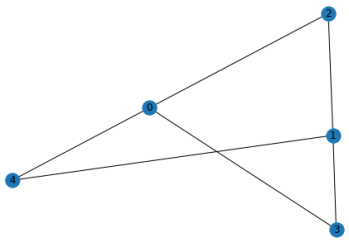
Experimental Results

Two example graphs are experimented with at the end of the Jupyter notebook.

```
def optimize_maxcut(graph, params, backend = Aer.get_backend('qasm_simulator'),
                    iterations = 2000, shots_per_iter = 1000):
    p = len(params)//2
    result = minimize(
        lambda angles: get_maxcut_total_objective(graph, angles, backend, n_shots = shots_per_iter)
                        params, method = 'COBYLA', options = {'maxiter': iterations}
    )
    alpha = result['x'][:p]
    beta = result['x'][p:]
    opt_circuit = get_maxcut_ansatz(graph, params)
    opt_counts = execute(opt_circuit, backend=backend, shots=iterations).result() \
                                                                        .get_counts(opt_circuit)
    opt_bitstring = max(opt_counts, key=opt_counts.get)
    return {'best_partition': opt_bitstring[::-1],
            'best_cut_value': -get_maxcut_singlerun_objective(graph, opt_bitstring[::-1]),
            'full_counts': opt_counts}
```



G1

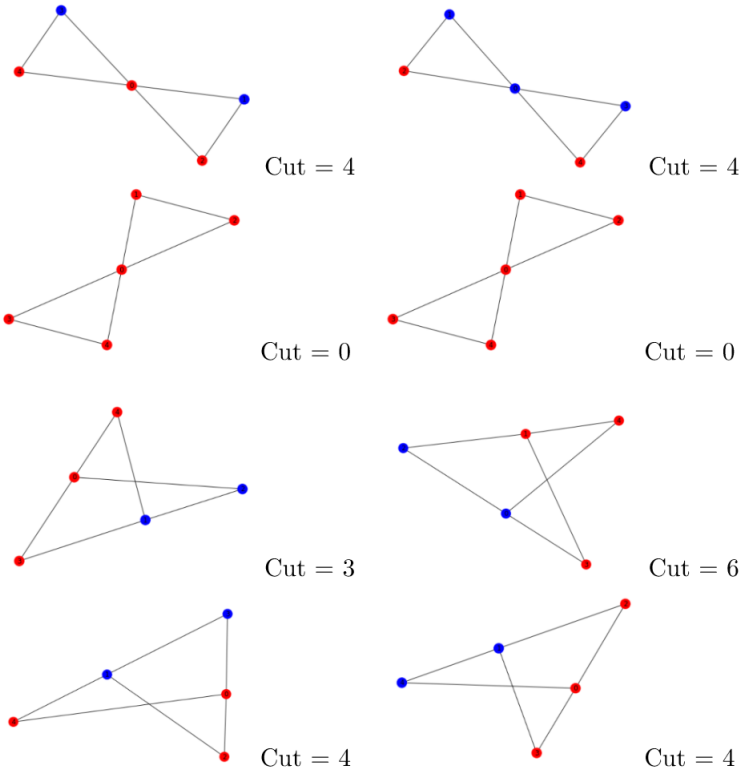


G2

Max-cut QAOA is run for two settings of p ($p = 1$ and $p = 2$) and on the QASM simulator or a real IBM quantum computer. Thus, there are eight total computations present in the notebook. In the table below, approximation ratios for all eight trials are presented (note that if the ratio is 1, then the optimal cut was found).

	G1		G2	
	p=1	p=2	p=1	p=2
QASM	**1**	**1**	**0.5**	**1**
IBM	**0**	**0**	**0.66**	**0.66**

As a sanity check, we apply the DRAW_MAXCUT_RESULT function on these graphs to show their partitions. The top four correspond to G1 and the bottom four to G2. All partitions match their calculated cut values.



Visualizations of graph partitions for G1 and G2

4 Conclusions and Connections

One trend I noticed was that QASM simulations performed definitively better than real quantum computers in approximating the max-cut solution on either graph. Holding the p value constant and comparing results for the same graph shows that running the circuit on IBM's quantum computers always does worse than the simulated answer. This is unsurprising. The QASM simulator is by definition error-free, since it does not truly manipulate qubits to achieve its computations. Rather, it simulates quantum behavior in a deterministic manner, with classical registers describing the state of the n-qubit system exactly. On the other hand, IBM's remote machine are prone to noise and have imperfect error mitigation, resulting in more overall noise in the data. This in turn reduces the prevalence of the correct partition occurring in the data.

Looking closely at the QASM result, it is clear that increasing the value of

p yields better results. This is again unsurprising, because the ansatz with more layers has more degrees of freedom to model the desired behavior. With QASM, there are no drawbacks to using one or two more layers, barring a slight increase in overall run-time; however, when running the circuits on IBM's quantum computers, increased depth is correlated with increased error, which is unacceptable given the relatively poor performance of the real machines on even $p = 1$ architectures.

Moreover, I think results would be improved if a better algorithm was chosen for the optimization routine. COBYLA is a general purpose, conventional optimization algorithm that is unaware of the innate randomness in quantum simulation results. When a measurement is made at the end of one of the circuit's runs during the optimization routine, there is a chance that the wrong state is measured, especially with the introduction of small phase errors in real quantum computers. These small errors may manifest themselves in the ultimate measurement, and COBYLA will take these results at face value. If we were able to incorporate some quantum understanding into COBYLA, providing it with the udnerstanding that not all measurements are 100% accurate, it may help the optimization converge faster and more accurately than it currently does.

Given more time, I might wish to implement some of the more recent research into QAOA that has occurred in the last decade to improve my results. In a paper by Shaydulin et al (2021)[4], the authors present a methodology for exploiting the inherent symmetry of some problems in order to acheive faster run-times. Since the main bottleneck in the QAOA variational cycle is on the quantum side rather than the classical side, this would be a major improvement and would allow me to experiment with larger, denser graphs. In another paper by Marwaha (2021)[3], it is suggested that there may not be the trivial linear increase in modeling capability that I presented as fact earlier in the paper when we increase p. Indeed, it appears that for some graphs, the max-cut ansatz works better with fewer layers. This would be another matter of great interest to investigate further.

References

1. Farhi et al, "A Quantum Approximate Optimization Algorithm". arXiv:1411.4028
2. Farhi et al, "Quantum Computation by Adiabatic Evolution". arXiv:quant-ph/0001106
3. Marwaha, "Local classical MAX-CUT algorithm outperforms p=2 QAOA on high-girth regular graphs". arXiv:2101.05513
4. Shaydulin et al, "Exploiting Symmetry Reduces the Cost of Training QAOA". arXiv:2101.10296