1. Quantum Approximation Optimization Algorithm (QAOA)

During my study, I found two excellent references [6,7] to get inducted by MaxCut and QAOA in general. Under the section II "III. QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM (QAOA)", subsection A: "The overview of QAOA" in the article from Choi and Kim [6] provides a didactic breakdown for QAOA. Incidentally, the breakdown is our crucial possession for destructing the MaxCut problem. Next, the article by Wang and Abdullah [7] provides *insightful run over* not only the algorithms but the circuits implementations too. **I firmly recommend you to read these two references.** Finally, the primary reference I used to produce this notebook is a Bachelor's thesis [8].

Quantum Approximate Optimization Algorithm (QAOA) is one of the flavor out of a collection termed as Variational Quantum Algorithms (VQA). QAOA is an approximation algorithm, that means it does not deliver the 'best' result, but only the 'good enough' result. How good of the result is quantified by a lower bound named approximation ratio. It runs on NISQ devices and are promising candidate for asserting the so-called "Quantum Supremacy" [3]. Farhi, E. et al.[2] introduced the algorithm, in their work they solved the Max-Cut problem. They went ahead to provided the approximation ratio for "3-Regular Graphs".

Modularity is a glaring presence in VQAs. So it is for QAOA, being an element of it. A variational algorithm comprises several modular components that can be readily combined, extended and improved with independents in quantum hardware and algorithms. The parts are, Objective function, Parametrized Quantum Circuit (PQC), Measurement and Parameter Optimization [1]. The last one is the only classical moving part to seek the best combination of parameter that maximizes the Objective function. Hence, the VQAs are dubbed as "*Hybrid Quantum-Classical Algorithms*" [4]. Another perspective to bear in mind for Max-Cut problem is Quadratic Unconstrained Binary Optimization (QUBO). This is a problem in mathematics and can be solved within it's domain. **Read more about QUBO here [5].**

To paraphrase the choi, et al. [6], the QAOA can be content by following overview.

A) Definition (Initial State). The initial state is as follows according to the superposition principle:

$$|s\rangle = |+\rangle^{\bigotimes n} = H^{\bigotimes n}|0\rangle$$

If $s \in \{0,1\}^n$ be the bit string then $|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle$

B) Definition II (Objective Function). In a combinatorial optimization problem defined on n-bit binary strings z, the objective function is defined as follows:

$$f(z) : \{0,1\}^n \to R$$

C) Definition III (Phase Operators). We can map the objective function, Definition I, to the phase-Hamiltonian, thus finding the optimal value of the objective function is a special case of finding the extremal eigenvalues for the phase Hamiltonian. The phase Hamiltonian $\hat{H}_P$ has eigenvalues that are the cost function for corresponding eigenstates. It acts diagonally on the computational basis states of 2^n dimensional Hilbert space (n-qubit space):

$$\hat{H}_p|z\rangle = f(z)|z\rangle$$

However, in the phase operators is unitary with γ parameter. This parameter is angle ϕ in the circuit implementation.

$$U_p(\gamma) = \exp(-i\gamma\hat{H}_p)$$

D) Definition IV (Mixing Operators). The mixing Hamiltonian H M is defined as follows:

$$\hat{H}_M = \sum_{j=1}^{n} \sigma_j^x$$

σ^x is the Pauli-X operator, it's usually used for bit flip operation. Similar to the phase operator, the mixing operator is defined as follows, with the parameter β represents the other spherical angle θ:

$$U_M(\beta) = \exp(-i\beta\hat{H}_M)$$

Based on these QAOA breakdown, the quantum state can be represented by two numbers (parameters) γ and β. So, there are p number of qubits in the problem then the circuit will have, at most, $2p$ number of parameters. And it is p-level circuit. Besides, you may have noticed the exponentiation of the two parameters, that is simply the evolution of quantum state. If the parameter were to be time, the exponential operator would be time evolution operator, and the operation would result into a time evolution of the quantum state.

2. A case of QAOA: Max-Cut (Introduction)

There are plethora of studies providing different formulation for the tid-bits like Objective function, Parametrized Quantum Circuit (PQC), and Optimization. Refer to Section II, "Building blocks of variational quantum algorithms" in the reference enlisted at [1]. However, for the Max-Cut problem things need not to be overly complicated. **The primary references [6,7] that I followed shows a straightforward way to solve the problem.**

There are different ways to express the Max-Cut problem. I have chosen to put it as following from the Wikipedia on MaxCut. The graphs I have dealt in this notebook are unweighted, if otherwise then the weights are calculated into the account.

> For a graph, a maximum cut is a cut whose size is at least the size of any other cut

So how is the "cut" counted? Consider a graph with m edges and n vertices. We seek the partition z of the vertices into two sets A and B which maximizes the clause below.

$$C(z) = \sum_{\alpha=1}^{m} C_\alpha(z)$$

$C_\alpha(z) = 1$ if z places one vertex from the α^{th} edge in set A and the other in set B. The vertices resides on either set A or set B so this welcomes qubits. Likewise, the qubit takes either 0 or 1 bitstring values. The vertice that is in A and in the other case is 1.

For a visualization let's checkout the graph with maximum cut. The graph is taken from the pennylane tutorial.

Figure 1 - A maximally cut graph.

Straight out of the box, the bitstring representing the graph is $|0101\rangle$ since the two vertices belongs to A and the other two belongs to B.

Let's start to be formal and put the mathematical definition of the problem below. I quoted it from [6]

> Given a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, maximize the number of edges crossing the cut. The cut separates $V1$ and $V2$, i.e., one vertex of the edges crossing the cut should be in $V1$ and the other vertex should be in $V2$, where $V1 \subset V$, $V1 \cup V2 = V$, and $V1 \cap V2 = \emptyset$.

3 Quantum implementation (Qiskit)

Qiskit provides high level abstraction to prepare and solve the MaxCut problem. However, I will attempt to decompose the algorithm and build a quantum circuit. Since it's a hybrid algorithm the classical optimizer will be put in action.

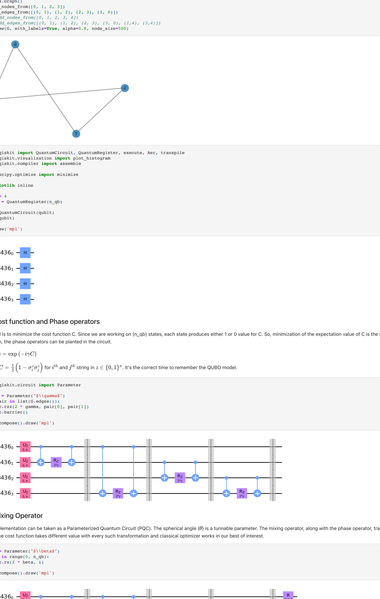


Figure 2 - Schematic for a Variational Quantum Algorithms (VQA). Refer [7] for the picture and further studies in VQA.

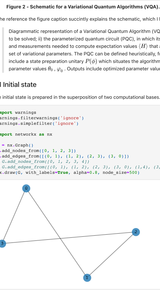
In the reference the figure caption succintly explains the schematic, which I have quoted below.

> Diagrammatic representation of a Variational Quantum Algorithm (VQA). A VQA workflow can be divided into four main components: i) the objective function $\hat{O}$ that encodes the problem to be solved, ii) the parameterized quantum circuit (PQC), in which its parameters θ are tuned to minimize the objective, iii) the measurement scheme, which performs the basis changes and measurements needed to compute expectation values $\langle \hat{H}_i^j \rangle$ that are used to evaluate the objective, and iv) the classical optimizer that minimizes the objective and proposes a new set of variational parameters. The PQC can be defined heuristically, following hardware-inspired ansatze, or designed from the knowledge about the problem Hamiltonian $\hat{H}$. It can also include a state preparation circuit acting $|\Psi_0\rangle$ which situates the algorithm to start in a particular region of parameter space. Inputs of a VQA are the circuit ansatz $U(\theta, \phi)$ and the initial parameter values θ_0 , ϕ_0 . Outputs include optimized parameter values θ_{opt} , ϕ_{opt} and the minimum of the objective.

3.1 Initial state

The initial state is prepared in the superposition of two computational bases.

In [48]:
```
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

import networkx as nx

G = nx.Graph()
G.add_nodes_from([0, 1, 2, 3])
G.add_weighted_edges_from([(0, 2, 3), (0, 3, 9)])
# G.add_edges_from([(0, 2), (0, 3)])
# G.add_edges_from([(0, 2), (2, 3), (3, 0), (3, 1),(1, 2),(1,0)])
nx.draw(G, with_labels=True, alpha=0.5, node_size=500)
```



In [49]:
```
from qiskit import QuantumCircuit, QuantumRegister, execute, Aer, transpile
from qiskit.visualization import plot_histogram
from qiskit.compiler import assemble

from scipy.optimize import minimize

%matplotlib inline

n_qb = 4
qubit = QuantumRegister(n_qb)

qc = QuantumCircuit(qubit)
qc.h(qubit)

qc.draw('mpl')
```

Out[49]:

3.2 Cost function and Phase operators

The goal is to minimize the cost function C. Since we are working on |n_qb⟩ states, each state produces either 1 or 0 value for C. So, minimization of the expectation value of C is the objective. For such aim, the phase operators can be planted in the circuit.

$$U(C, \gamma) = \exp(-i\gamma C)$$

Where, $C = \frac{1}{2}\left(1 - \sigma_j^z\right)$ for i^{th} and j^{th} string in $z \in \{0,1\}^n$. It's the correct time to remember the QUBO model.

In [50]:
```
from qiskit.circuit import Parameter

gamma = Parameter('$\gamma$')
for pair in list(G.edges()):
    qc.rzz(2 * gamma, pair[0], pair[1])
    qc.barrier()

qc.decompose().draw('mpl')
```

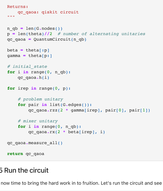
Out[50]:

3.3 Mixing Operator

This implementation can be taken as a Parametrized Quantum Circuit (PQC). The spherical angle (θ) is a tunnable parameter. The mixing operator, along with the phase operator, transforms the initial state. The cost function takes different value with every such transformation and classical optimizer works in our best of interest.

In [51]:
```
beta = Parameter('$\theta$')
for qb in range(n_qb):
    qc.rx(2 * beta, qb)

qc.decompose().draw('mpl')
```

Out[51]:



3.4 Classical Optimizer

From the Fig 2. it's clear that the job for the optimizer is to check for the minimum expectation value of the cost function. At the end of every execution, it will prescribe a new set of parameters (β, θ). This new parameters set will be fed into the "PQC" for next iteration of cost function. And it's basically *rinse and repeat* until the minimum value of the cost function is obtained.

Classical Machine Learning models have myriad of optimizer. However, by far the most popular is "Adam" optimizer. For this particular task, I won't go with it and rather choose "COBLYA" optimizer. It is a *numerical optimization method for constrained problems where the derivative of the objective function is not known.* The reason to choose it over "Adam" is pretty much self-explanatory now, isn't it?

It's easy to see the cost function C is determined by two adjacent vertices (say i^{th} and j^{th}). The cut will split the vertices into two sets and likewise the edges, if both are in same set then C_{ij} bears 0 or else 1. So a simpler way to put the cost function will be as follows.

$C_{i,j}(z) = (1 - \delta_{z_i z_j})$, where, δ_{ij} is the Kronecker delta function.

For a better understanding of this simplified form of "Cost function" visit page 4,3 and page 44 in the reference [8].

The cost function value will be sum amongst all the adjacent vertices indexed by $\{i, j\}$. The expectation value will be amongst all the bit string in the set $\{0,1\}^n$.

In [52]:
```
from qiskit.compiler import assemble
import sys
def cost_function(theta):
    cost = 0
    cumulative = 0
    sum_count = 0

    backend = Aer.get_backend('qasm_simulator')
    qqc = create_qaoa_circ(theta)
    result = execute(qqc, backend, shots=1024).result().get_counts()

    for bitstring, count in result.items():
        for i in list(G.edges()):
            if bitstring[i[0]] != bitstring[i[1]]:
                cost -= 1
        cumulative += cost * count
        sum_count += count

    return cumulative/sum_count
```

The above code snippet provides the expectation value, $\langle C \rangle$. The function takes in one argument, θ, which the mixing parameter. Moreover, there's a PQC generated by the function "create_qaoa_circ". I have built the parameterized circuit all along this notebook, however, it can only be measured once before it is lost for forever, rendering it good for nothing. During the optimization, the PQC will be tortured by measurement at the end of every iteration. Hence, the PQC should be wrapped around by a function, such that, its return value is the PQC. The qiskit has provided the code snippet [9].

In [53]:
```
def create_qaoa_circ(theta):
    '''
    Creates a parametrized qaoa circuit

    Args:
        G: networkx graph
        theta: list
               unitary parameters

    Returns:
        qc: qaoa: qiskit circuit
    '''

    n_qb = len(G.nodes())
    p = len(theta)//2  # number of alternating unitaries
    qc_qaoa = QuantumCircuit(n_qb)

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

    # initial_state
    for i in range(0, n_qb):
        qc_qaoa.h(i)

    for irep in range(0, p):

        # problem unitary
        for pair in list(G.edges()):
            qc_qaoa.rzz(2 * gamma[irep], pair[0], pair[1])

        # mixer unitary
        for i in range(0, n_qb):
            qc_qaoa.rx(2 * beta[irep], i)

    qc_qaoa.measure_all()

    return qc_qaoa
```

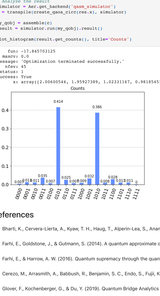
3.5 Run the circuit

It's now time to bring the hard work in to fruition. Let's run the circuit and see for the results. There must be two quantum states which minimizes the cost function, $|0101\rangle$ and $|1010\rangle$.

In [54]:
```
dummy = lambda x: expectation(x)
res = minimize(dummy,
                       [1.0, 1.0, 1.0, 1.0],
                       method='COBYLA')

print (res)

# Analyse the result
simulator = Aer.get_backend('qasm_simulator')
qc = transpile(create_qaoa_circ(res.x), simulator)

qq_obj = assemble(qc)
result = simulator.run(qq_obj).result()

plot_histogram(result.get_counts(), title='Counts')
```

```
     fun: -1.8457031250000005
   maxcv: 0.0
 message: 'Optimization terminated successfully.'
    nfev: 81
  status: 1
 success: True
       x: array([2.03600546, 1.95927389, 1.02331167, 0.90195451])
```

Out[54]:



References

1. Bharti, K., Cervera-Lierta, A., Kyaw, T. H., Haug, T., Alperin-Lea, S., Anand, A., ... & Aspuru-Guzik, A. (2021). Noisy intermediate-scale quantum (NISQ) algorithms. arXiv preprint arXiv:2101.08448.
2. Farhi, E., Goldstone, J., & Gutmann, S. (2014). A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028.
3. Farhi, E., & Harrow, A. W. (2016). Quantum supremacy through the quantum approximate optimization algorithm. arXiv preprint arXiv:1602.07674. Chicago
4. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., ... & Coles, P. J. (2021). Variational quantum algorithms. Nature Reviews Physics, 1-20. Chicago
5. Glover, F., Kochenberger, G., & Du, Y. (2019). Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models. 4OR, 17(4), 335-371.
6. Choi, J., & Kim, J. (2019, October). A tutorial on quantum approximate optimization algorithm (qaoa): Fundamentals and applications. In 2019 International Conference on Information and Communication Technology Convergence (ICTC) (pp. 138-142). IEEE.
7. Wang, Q., & Abdullah, T. (2018). An introduction to quantum optimization approximation algorithm.
8. Guerreschi, G., 2020. Solving Combinatorial Optimization Problems using the Quantum Approximate Optimization Algorithm. [online] Scholar.afit.edu. Available at: https://scholar.afit.edu/cgi/viewcontent.cgi?article=4284&context=etd.
9. Qiskit.org. n.d. Solving combinatorial optimization problems using QAOA. [online] Available at: https://qiskit.org/textbook/ch-applications/qaoa.html [Accessed 16 September 2021].