Exploring Knapsack problem with VQE, QAOA using Qiskit

Team EQ

Juon Kim @rkfqns13

1. What is Knapsack problem?

Knapsack problem is a constrained combinatorial optimization problem that refers to the general problem of packing a knapsack with the most valuable items without exceeding its weight limit. To sum up, it is a NP-complete problem that finds high values within a limited weight.

For example, let's assume that there are 4 items. First item's value is 120 and weight is 13. Second item's value is 54 and weight is 6. Third item's value is 28 and value is 15. The last item's value is 49 and weight is 9. If the max weight is 32, this knapsack problem answer is to pick first, second, and last items. The max value is 223.

The mathematical representation of knapsack problem is as follows. In the 0/1 knapsack problem, there are *n* items, and each item's weight *w* and value *c*. The sum of the weight of the entire item does not exceed *W*.

$$\sum_{\alpha=1}^{n} w_\alpha x_\alpha \leq W$$

2. Knapsack problem to Ising model

The Ising model is a quadratic model with binary variables without restrictions. The model consists of discrete variables that represent magnetic dipole moments of atomic spins that can be in one of two states. So it is used to phase transitions and certain properties of particles in a physical system that evolves over time. It is also NP-hard problems.

The mathematical representation of ising model is as follows. Each spin has value *s* and there are external forces *h* and respective forces *J*. Hamiltonian H is sum of all vertices and the interacting forces on the edges of the graph.

$$H(s) = \sum_{i}^{n} h_i s_i + \sum_{i}^{n} \sum_{j>i}^{n} J_{ij} s_i s_j.$$

By applying this concept to knapsack problem, the solution is H.

$$H = H_A + H_B;$$

$$H_A = A\left(1 - \sum_{n=1}^{W} y_n\right)^2 + A\left(\sum_{n=1}^{W} n y_n - \sum_{\alpha} w_\alpha x_\alpha\right)^2$$

$$H_B = -B\sum_{i=1}^{N} c_i x_i.$$

3. Knapsack problem with VQE

Variational Quantum Eigensolver algorithm(VQE) is a hybrid algorithm that uses a variational technique and interleaves quantum and classical computations in order to find the minimum eigenvalue of the Hamiltonian H of a given system. VQE is suitable for noisy intermediate scale quantum computation.

The process of solving the knapsack problem with VQE is as follows.

1) Make pauli list
2) Make matrix with values
3) Calculate the coefficient Weights * Values
4) Append the matrix and coefficient in pauli list
5) Make VQE algorithms
6) Run the VQE algorithm and extract the highest freqquency shot

Detailed code descriptions are as follows.

```
import matplotlib.pyplot as plt
import matplotlib.axes as axes
%matplotlib inline
import numpy as np

from qiskit import BasicAer
from qiskit.tools.visualization import plot_histogram
from qiskit.aqua.algorithms import VQE, ExactEigensolver
from qiskit.aqua.components.optimizers import SPSA
from qiskit.aqua import QuantumInstance

from qiskit.quantum_info import Pauli
from qiskit.aqua.operators import WeightedPauliOperator
from collections import OrderedDict
import math
```

Import the useful packages.

```python
def get_knapsack_operator(values, weights, max_weight):

    y_size = int(math.log(max_weight, 2)) + 1
    n = len(values)
    num_values = n + y_size
    pauli_list = []
    shift = 0

    for i in range(n):
        coefficient = weights[i] * values[i]

        xp = np.zeros(num_values, dtype=np.bool)
        zp = np.zeros(num_values, dtype=np.bool)
        zp[i] = not zp[i]
        pauli_list.append([coefficient, Pauli(zp, xp)])
        shift -= coefficient

    return WeightedPauliOperator(paulis=pauli_list), shift
```

Make the get_knapsack_operator function. Here, make pauli list and declare variable shift, which determines the angle of ry rotation. The coefficient is each item's weight*value.

Then make two zero matrices with values and apply not gate to one matrix. When the whole process is over, append the coefficient and two matrices in pauli list. Shift value subtract the coefficient value and repeat these process for every items.

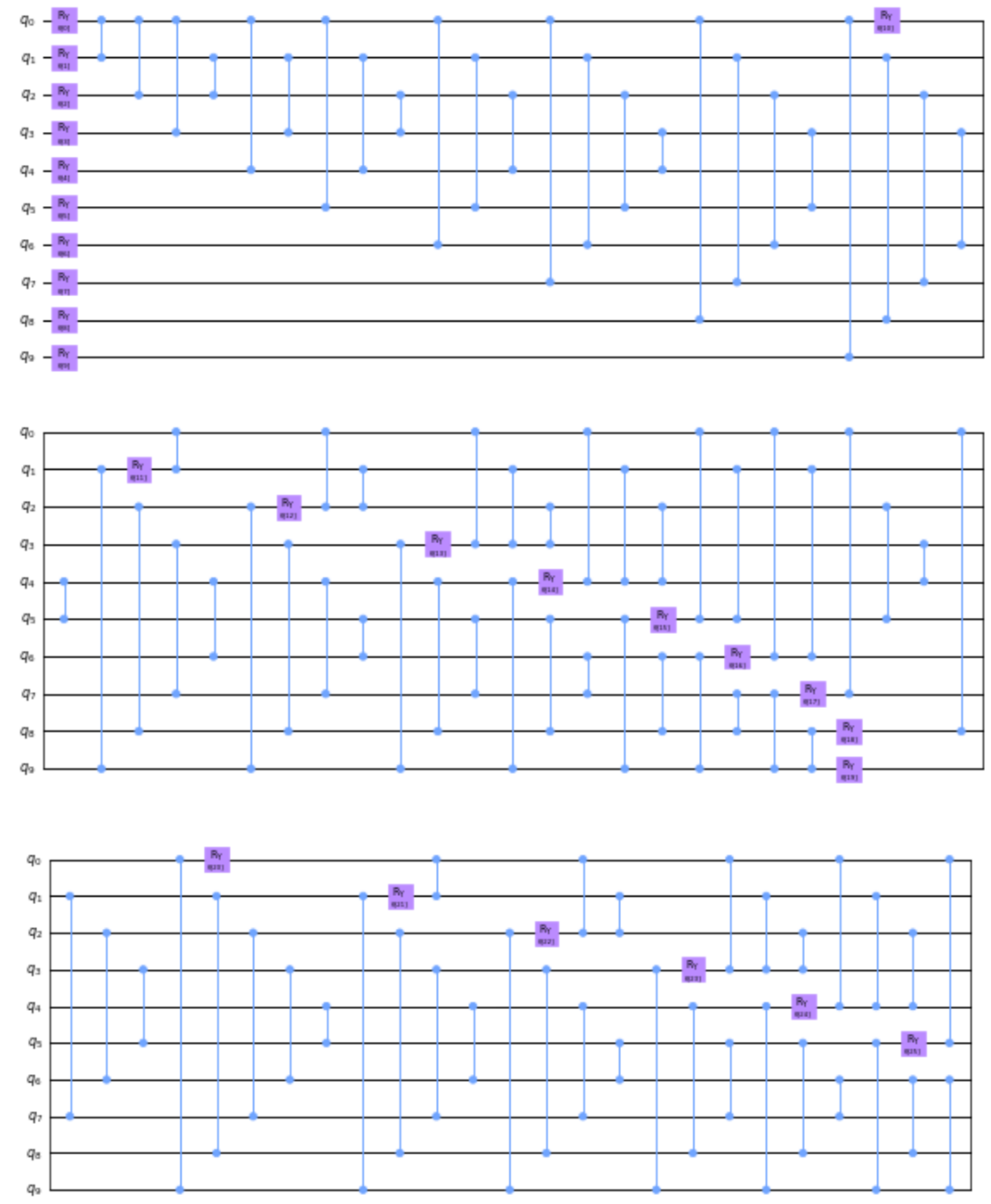
```python
values = [120, 54, 28, 49]
weights = [13, 6, 15, 9]
w_max = 32

qubitOp, offset = get_knapsack_operator(values, weights, w_max)
```

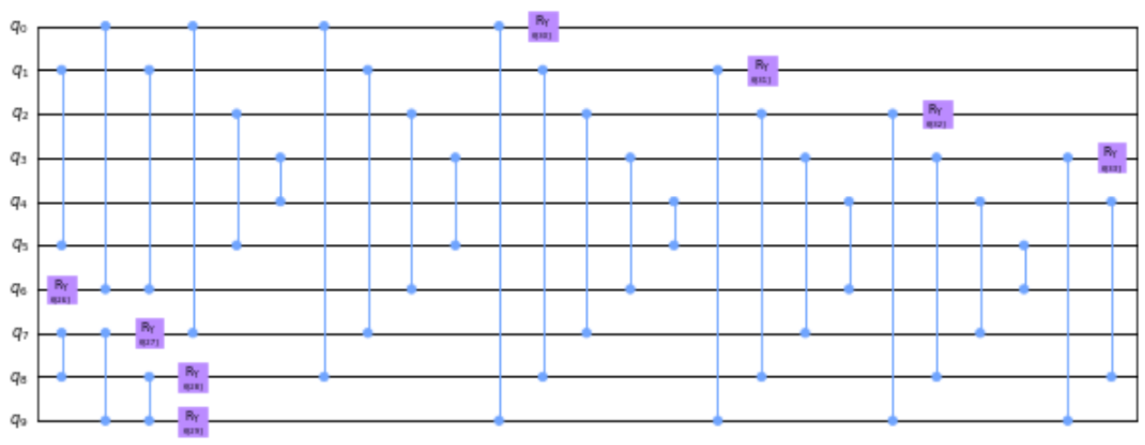
Put the values and weights of each item in the array and specify the maximum weight value. Here, I put the same items as the example in the first page. Then I apply get_knapsack_operator function with values, weights and w_max and named it qubitOp.

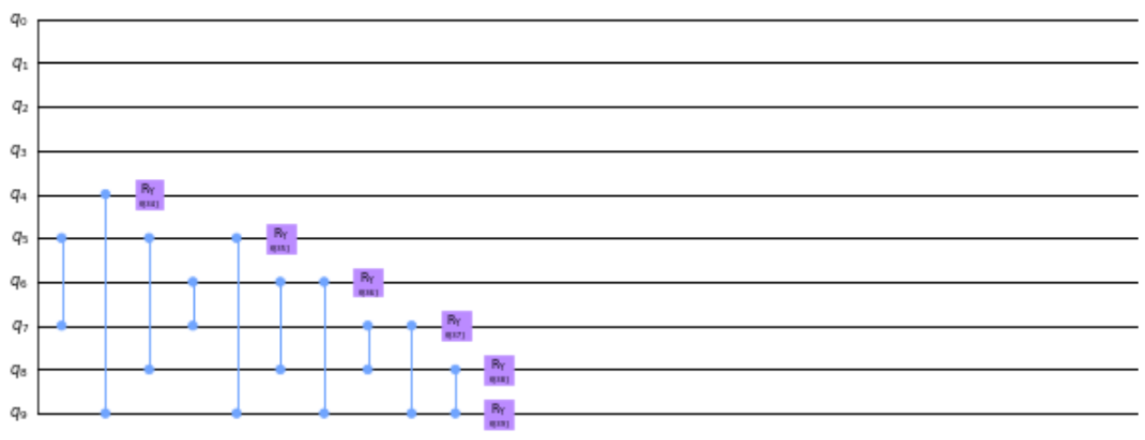
```python
seed = 10000

spsa = SPSA(max_trials=100)
ansatz = TwoLocal(rotation_blocks='ry', entanglement_blocks='cz')
vqe = VQE(qubitOp, ansatz, spsa)

backend = provider.get_backend('ibmq_qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=100, seed_simulator=seed, seed_transpiler=seed)

result_shots = vqe.run(quantum_instance)

most_lightly_shots = result_shots['eigvecs'][0]
t = []
for k, v in most_lightly_shots.items():
    t.append((k,v))
```

Then make the VQE algorithm and run it. I use simultaneous perturbation stochastic approximation(spsa) optimizer, which is an algorithmic method for optimizing systems with multiple unknown parameters. I put the number of max trials 100.

TwoLocal is a variety of forms, a parameterized circuit that can vary, and a classic optimizer SLSQP. They are created as separate instances and passed to VQE when configured. I use ry rotation blocks and cz entaglement blocks. The model I made TwoLocal named ansatz looks like this:







Using qubitOp, ansatz, and spsa optimizer, I make VQE algorithm. I used 10 qubits, so I tried to use ibmq_16_melbourne. But maybe because of the limitations of quantum computers, it spends a lot of time to run vqe. Thus I use qasm simulator to run the vqe algorithm. Then I choose the minimum eigenvector.

```
maxcount=0
for i in range(len(t)):
    a = t[i][1]
    if maxcount<a:
        maxcount=a

for i in range(len(t)):
    if t[i][1] == maxcount:
        shot=t[i][0]

print(shot[:len(values)])
```

```
1101
```

Lastly, I extract the highest frequency shot. The result ‘1101’ means first, second, and fourth items are selected from above example.

4. Knapsack problem with QAOA

Quantum Approximate Optimization Algorithm(QAOA) is a general technique that can be used to find approximate solutions to combinatorial optimization problems, in particular problems that can be cast as searching for an optimal bitstring.

The process of solving the knapsack problem with QAOA is as follows.

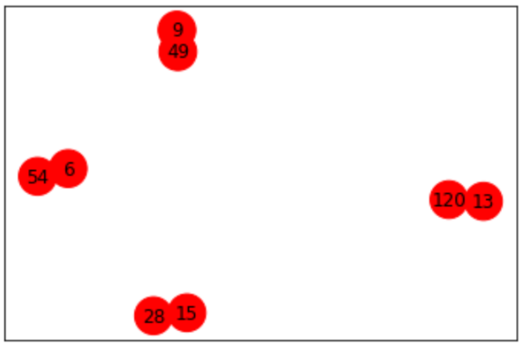
1) Make the graph
2) Grid search for the minimizing variables
3) Make a quantum circuit
4) Apply hadamard gates to all qubits
5) Run the QAOA algorithm
6) Extract the highest frequency shot

```
import networkx as nx
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile, assemble
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram
```

Import the useful packages.

```
n = 4
V = [120, 54, 28, 49]
E = [(120,13,1.0),(54,6,1.0),(28,15,1.0),(49,9,1.0)]

G = nx.Graph()
G.add_nodes_from(V)
G.add_weighted_edges_from(E)

default_axes = plt.axes(frameon=True)
pos          = nx.spring_layout(G)

nx.draw_networkx(G, node_color=colors, node_size=600, alpha=1, ax=default_axes, pos=pos)
```

Make the graph. For vertex, I put values and for edges, I put values and weights set. And add the vertex and edge in graph. The graph looks like this:



```
step_size   = 0.1;

a_gamma          = np.arange(0, np.pi, step_size)
a_beta           = np.arange(0, np.pi, step_size)
a_gamma, a_beta = np.meshgrid(a_gamma,a_beta)

F1 = 3-(np.sin(2*a_beta)**2*np.sin(2*a_gamma)**2-0.5*np.sin(4*a_beta)
        *np.sin(4*a_gamma))*(1+np.cos(4*a_gamma)**2)

result = np.where(F1 == np.amax(F1))
a      = list(zip(result[0],result[1]))[0]

gamma  = a[0]*step_size;
beta   = a[1]*step_size;
```

By qiskit QAOA guideline, solving combinatorial optimization problems using QAOA results that the expectation value is:

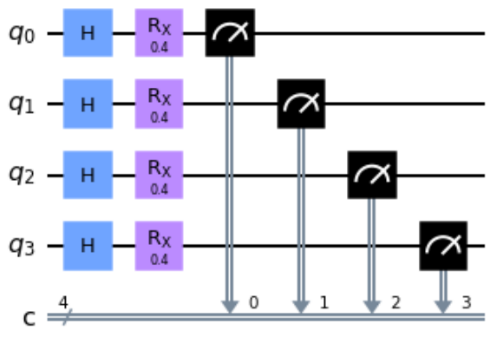
$$F_1(\gamma,\beta) = 3 - \left(sin^2(2\beta)sin^2(2\gamma) - \frac{1}{2}sin(4\beta)sin(4\gamma)\right)\left(1 + cos^2(4\gamma)\right)$$

So calculate F1 using a_gamma and a_beta to grid search for the minimizing variables.

```
QAOA = QuantumCircuit(len(V), len(V))

QAOA.h(range(len(V)))

for edge in E:
    k = edge[0]
    l = edge[1]

QAOA.rx(2*beta, range(len(V)))
QAOA.measure(range(len(V)),range(len(V)))

QAOA.draw('mpl')
```

Make a quantum circuit and apply hadamard gates to all qubits. Then apply the rx gates with angle beta to all qubits and measure the QAOA circuit.

The QAOA circuit looks like this:



```
shots = 1000

TQAOA = transpile(QAOA, backend)
qobj = assemble(TQAOA, shots=shots)
QAOA_results = backend.run(qobj).result()
```

```
import operator
circ={}
circ=QAOA_results.get_counts()

a=max(circ.items(), key=operator.itemgetter(1))[0]
b=max(circ.items(), key=operator.itemgetter(1))[1]
print(a)
print(b)
```

```
1101
76
```

Run the QAOA algorithm and extract the highest frequency shot. The result ‘1101’ means first, second, and fourth items are selected from above example.

5. Conclusion

There are some differences between VQE and QAOA algorithm. VQE requires (number of items + 1) * 2 qubits but QAOA requires number of items qubits. VQE uses minimum eigenvalue of H but QAOA uses grid search for the minimizing variables. VQE uses Ry gates but QAOA uses Rx gates. VQE uses pauli operator but QAOA uses graph. However, the two algorithms are the same in that they build and measure circuits and determine the most frequent value. In summary, it is as shown in the table below.

	VQE	QAOA
Idea	minimum eigenvalue of H	grid search for the minimizing variables
Tools	pauli operator	graph
Qubits	(number of items + 1) * 2	number of items
Gates	Ry	Rx