Quantum TSP

TSP Group:
Yifan Zuo,
JiaQing Hu,
Mingzhen Mou,
Jiangsheng Guo,
Zhongliang Wang

Introduction

- TSP(Travelling Salesman Problem):
  - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?”(wiki)
  - It is an NP-hard problem in combinatorial optimization.
  - In our project, the distance between cities are symmetric.
  - In our project, the first and last cities are not defined.
- Classical VS Quantum:
  - Classical brute force: number of possible solutions is growing exponentially with each additional city.
  - Quantum TSP: polynomial speed up.

Prerequisites

- Forest SKD:
  - In order to connect to Quantum Virtual Machine.
- Pyquil:
  - Use pyquil.api, pyquil.paulis, pyquil.gates.
- Grove:
  - QAOA: Quantum approximate optimization algorithm.
  - QAOA is an algorithm for solving a broad range of optimization problems using NISQ (Noisy Intermediate-Scale Quantum) devices.

QAOA

- QAOA:
  - An algorithm that combine quantum part and classical part.
  - The quantum part prepare the quantum state and measure it , repeat the process.
  - The classical part use Nelder-Mead algorithm find the most improved angles iteratively.
- betas, gammas = QAOA_inst.get_angles()
  - Beta and gammas are the angles that we got.
- most_common_result, _ = QAOA_inst.get_string(betas, gammas, samples=50000)
  - By knowing the angles, we can get the most common results. Since the result is probabilistic, we should find the most common result.

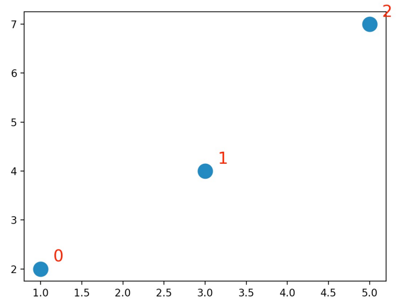
Problem representation

- Graph:
  - For example, we want to present 3 cities: [(1,2), (3,4), (5,7)]
  - The distance matrix is:

    [[0, 2.82842712, 6.40312424]

    [2.82842712 0. 3.60555128]

    [6.40312424 3.60555128 0.]]
  - Cost function is the sum of the cost we want to minimize.



Encode Problem

- Points representation:
  - [0, 1, 2, 3]: Means city 0 to 1 … to 3.
- Time-city matrix:
  - row represents time slots,
  - columns represent cities
  - Point representation and time-city matrix are interchangeable.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Solving TSP

- Naïve Approach
  - Start with 3 cities, calculate the best distance(fig 1.1)
  - Then go for more cities…
  - Use *ForestTSPSolverNaive()* to solve the whole problem.
    - Naïve Solution:[0, 1, 2, 2]

```
distance
 [[0.  4.  5.]
 [4.  0.  3.]
 [5.  3.  0.]]
```

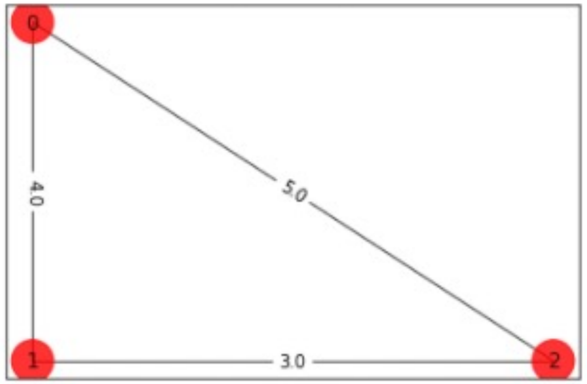


Fig 1.1

Solving TSP

- Naïve Approach Contd.
  - The output doesn't make sense(fig 2.1)
  - One big reason is that *ForestTSPSolverNaive()* doesn't have enough constraints. – Adding penalties to the solver

```
print("Your binary solution is:", tsp_solver.most_frequent_string)
print("My initial binary solution is:", (1,1,1,0,0,0,0,0,1))
```

```
Your binary solution is: (1, 0, 0, 1, 0, 0, 0, 0, 1)
My initial binary solution is: (1, 1, 1, 0, 0, 0, 0, 0, 1)
```

(fig 2.1)

Solving TSP

- Improved naïve approach
  - Create penalty matrices - add huge penalty for the states we don't want.
  - Example of 3 cities
    - At t = 0, only possible states are 100,010,001. So we penalize other states by creating an operator matrix(fig 3.1), such operator ignores 100,010,001 and 111 and penalizes the rest states.
    - To penalize state 111, it requires a little bit of linear algebra.
    - After we combine those operators to the final operator, we can

    implement it in QAOA.

Our solution will be the following vector state:
\ket$\psi_{sol}$ = \ket$q_1q_2q_3q_4q_5q_6q_7q_8q_9$

```
Z1Z2Z3 = np.kron(Z,np.kron(Z,Z))
print("Product of Z1, Z2 and Z3")
print(Z1Z2Z3)
print("Our operator")
basic_penalty = 0.5 * (np.eye(8) + Z1Z2Z3)
print(basic_penalty)
```

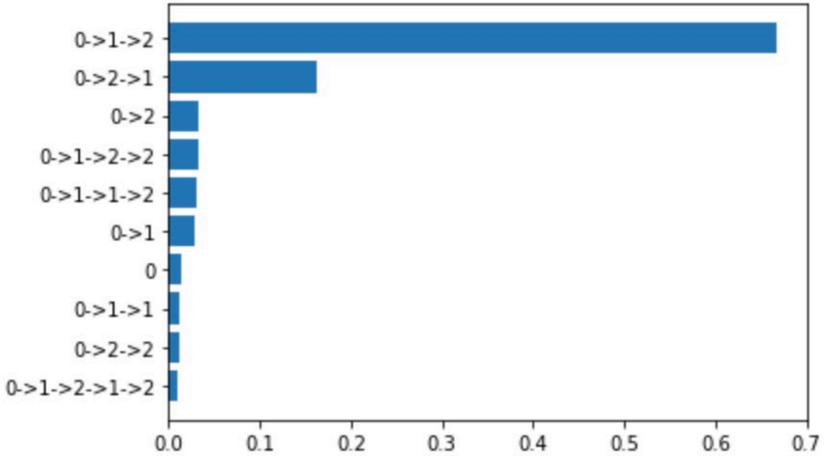
```
Product of Z1, Z2 and Z3
[[ 1  0  0  0  0  0  0  0]
 [ 0 -1  0  0  0  0  0  0]
 [ 0  0 -1  0  0  0  0  0]
 [ 0  0  0  1  0  0  0  0]
 [ 0  0  0  0 -1  0  0  0]
 [ 0  0  0  0  0  1  0  0]
 [ 0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0 -1]]
Our operator
[[1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

Fig 3.1

Solving TSP

- Improved naïve approach:
  - Results:

Best order from brute force = (0, 1, 2) with total distance = 7.0



Solving TSP

- Naïve approach summary
  - QAOA - An algorithm that combine quantum part and classical part.
  - Initially not good enough to solve TSP due to the missing of constraints.
  - Encoding constraints and costs in QAOA to solve TSP.
- Possible improvements
  - Tuned parameters
  - More compact encoding
  - Check for changes dynamically

Implement using Qiskit

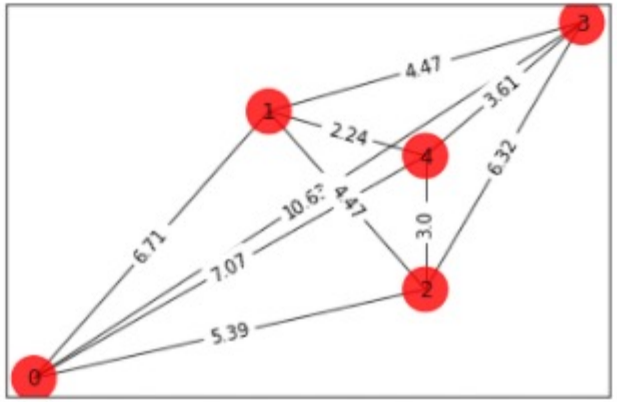
useful packages

```
# useful additional packages
import matplotlib.pyplot as plt
import matplotlib.axes as axes
%matplotlib inline
import numpy as np
import networkx as nx

from qiskit import Aer
from qiskit.tools.visualization import plot_histogram
from qiskit.circuit.library import TwoLocal
from qiskit.optimization.applications.ising import max_cut, tsp
from qiskit.aqua.algorithms import VQE, NumPyMinimumEigensolver
from qiskit.aqua.components.optimizers import SPSA
from qiskit.aqua import aqua_globals
from qiskit.aqua import QuantumInstance
from qiskit.optimization.applications.ising.common import sample_most_likely
from qiskit.optimization.algorithms import MinimumEigenOptimizer
from qiskit.optimization.problems import QuadraticProgram

# setup aqua logging
import logging
from qiskit.aqua import set_qiskit_aqua_logging
```

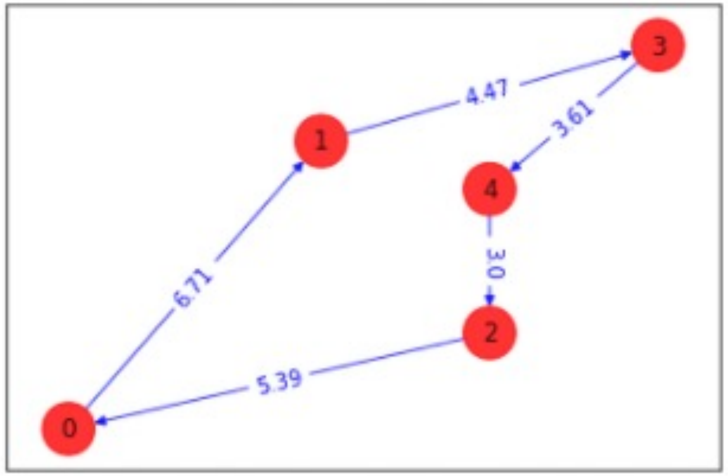
Implement using Qiskit

- Example: 5 cities [[0,0],[3,6],[5,2][7,8],[5,5]]
- distance matrix and graph(right):
- result：

```
distance
 [[ 0.    6.71  5.39 10.63  7.07]
 [ 6.71  0.    4.47  4.47  2.24]
 [ 5.39  4.47  0.    6.32  3.  ]
 [10.63  4.47  6.32  0.    3.61]
 [ 7.07  2.24  3.    3.61  0.  ]]
```




```
Best order from brute force = (0, 1, 3, 4, 2) with total distance = 23.18
```



Reference

- Max-Cut and Traveling Salesman Problem. From https://qiskit.org/documentation/tutorials/optimization/6_examples_max_cut_and_tsp.html
- Quantum_tsp_tutorials. From https://github.com/mstechly/quantum_tsp_tutorials

Thanks ！