A Quantum Phase Estimation Approach to the Traveling Salesman Problem

Vik Pattabi
vpattabi@stanford.edu

June 6, 2019

1 Introduction

The Traveling Salesman Problem (TSP) asks the following: "Given a network representing a list of cities and with edges weighted as the distance between each pair of cities (if connected), what is the shortest possible route that visits each city exactly once and returns home?". This NP-hard problem is a classic question in combinatorial optimization. Although a large number of heuristic and exact approaches are known, it remains unproven that a solution can be found for TSP that runs in even $O(1.9999^n)$ [5].

Consequently, emerging quantum optimization algorithms seem promising in the context of designing more efficient TSP solvers. One such approach is [1]'s notion of a quantum approximate optimization algorithm. [2] takes this idea further with the quantum alternating operator ansatz (QAOA) which yields more families of operators than those formulated in [1]'s work. Unfortunately, these algorithms necessitate at minimum n^2 qubits given a world with n cities to construct the QAOA mapping; even on extremely small worlds (such as size 4 or 5), this is an intractable number of qubits.

By contrast, [4] posits a quantum phase estimation approach to TSP. Applying the quantum phase estimation algorithm on specific eigenstates which represent combinations of distance-phases builds a mapping from Hamiltonian cycles in the network to oath lengths easily allowing for path length minimization. Furthermore, this algorithm requires less than n^2 qubits. This paper details an implementation of the quantum phase estimation approach in [4] and presents results and analysis of running this TSP solver on various 4-node networks.

2 Algorithm

The quantum phase estimation algorithm (also referred to as quantum eigenvalue estimation) can be used to estimate the eigenvalue (or phase) of an eigenvector of a unitary operator. Based on work by [4], I model the distances between cities as phases by transforming the city network's adjacency matrix. I then construct unitary operators whose eigenvalues are various combinations of these phases such that each combination represents a Hamiltonian cycle on the original network. Finally, selecting the eigen-state corresponding to the minimum eigenvalue yields the path of lowest cost in the network. The specifics of each step in this process are below.

2.1 Setting up the unitary operators

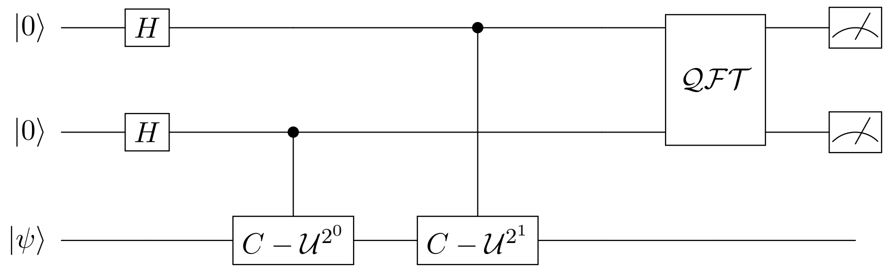
Given an adjacency matrix of size $4x4$ such as:

$$A = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \phi_{1,4} \\ \phi_{2,1} & \phi_{2,2} & \phi_{2,3} & \phi_{2,4} \\ \phi_{3,1} & \phi_{3,2} & \phi_{3,3} & \phi_{3,4} \\ \phi_{4,1} & \phi_{4,2} & \phi_{4,3} & \phi_{4,4} \end{bmatrix} \tag{1}$$

we can construct a phase matrix P such that $P_{a,b} = e^{i\phi_{a,b}}$. Then, we construct 4 $4x4$ unitary operators U_j from P such that $(U_j)_{k,k} = [P]_{j,k}$. As noted in [4], the tensor product of the n U_j operators $\mathcal{U} = U_1 \otimes U_2 ... \otimes U_n$ will also be a unitary matrix, as each of its components is a diagonal unitary matrix.

2.2 Quantum Phase Estimation

The quantum phase estimation (QPE) circuit operates on two registers of qubits. The lower register is repeatedly operated on by a controlled-$\mathcal{U}$ operator controlled by qubits in the upper register; an inverse quantum fourier transform is then applied to the upper register qubits which serve as the estimator qubits for an n-bit estimate of the phase θ. Equation 2 presents a sample of this circuit operating on eigenvector $|\psi\rangle$ using 2 qubits for phase estimation.



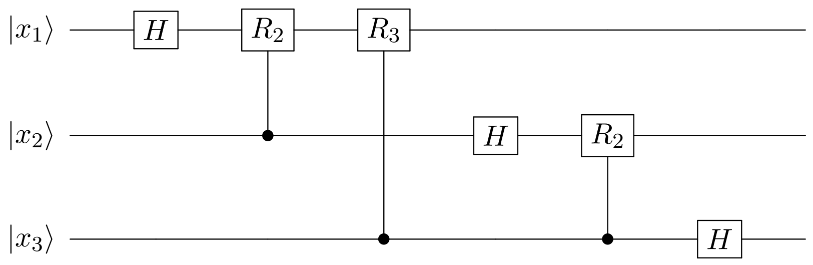
(2)

2.3 The Quantum Fourier Transform

The final step of QPE is the Quantum Fourier Transform (QFT), a quantum analog to the inverse discrete Fourier transform. The QFT circuit uses Hadamard and controlled-phase shift gates where the phase gate in the latter case is defined by [3] as:

$$R_m = \begin{bmatrix} 1 & 0 \\ 0 & \omega_m \end{bmatrix} \quad \text{where} \quad \omega_m := e^{2i\pi/2^m} \tag{3}$$

Equation 4 demonstrates an implementation of this circuit on 3 qubits (indexed from $i \in 1, 2, 3$). Although this is only half the number of estimator qubits used for the experiments ran in this paper, the QFT circuit is easily generalizable. Specifically, $(n - i)$ R_m gates are applied on a given qubit i beginning with $m = 2$ and continuing to $m = (n - i)$.



(4)

2.4 Benefits over QAOA

Unlike the QAOA approach which requires n^2 qubits, this approach only requires $n \log_2 n$ qubits for characterizing the eigenstate. Specifically, each path consists of n qubits represented by $\log_2 n$ bits. Consequently, the total number of qubits required for the QPE approach is $n \log_2 n + k$ where k is the number of qubits used for estimation.

3 Data

Due to the proof-of-concept nature of this project, only small networks were used to evaluate the implemented algorithm. Specifically, symmetric fully-connected networks of 4 nodes were used for this experiment. An example of one such network can be found in Figure 1.

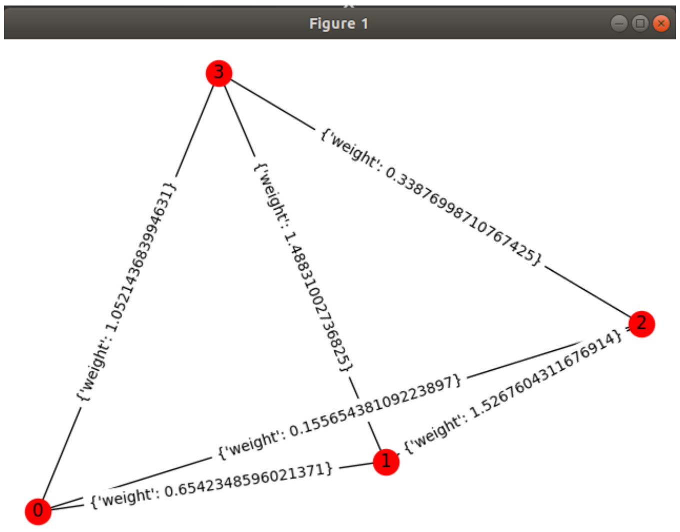


Figure 1
3
2
1
0
{'weight': 1.052143683994631}
{'weight': 1.48831002736825}
{'weight': 0.33876998710767425}
{'weight': 0.15565438109223897}
{'weight': 1.5267604311676914}
{'weight': 0.6542348596021371}

Figure 1: A sample graph randomly generated by `data.py`. Note that all edge weights are less than $\pi/2$ to limit the maximum phase value.

The network need not be fully connected. However, as the network's adjacency graph informs the creation of the $\mathcal{U}$ operator, representing the absence of edges in the adjacency matrix is tricky, which is why this paper focused on fully connected networks. One possible technique to do so sets all non-existent edge indices in the matrix to be $\pi/2$ (the maximum allowed edge weight given a 4-node network in order to limit the total phase as less than 2π).

4 Implementation Details

4.1 Source Code

My implementation of this algorithm (along with other project-relevant code) can be found here or on my GitHub at `https://github.com/vikpattabi/QuantumTSP`. The repository includes the core `solver.py` and three helper modules (`data`, `tsp_funcs` and `quantum_funcs`) which implement helper functions for generating sample graphs, creating the unitary operators, and building QUIL programs to be run on a QVM. The repository README includes more information about setting up and running the project.

4.2 Gate Implementations

[4] implements the above circuits using the $u1$ and r_m gates native to IBM's quantum experience architecture. By contrast, I elected to use pyQuil's `defGate` command to define these gates from numpy matrices

as can be seen in `quantum_funcs.py`. Furthermore, I used Hadamard gates to entangle the estimator qubits at the start of each program and X gates on select state qubits to 'encode' specific eigen-states to test. This design choice became problematic when attempting to incorporate a noise model and analyze its effects.

4.3 Inspecting Possible Eigenstates

Given the use of a fully-connected 4-node network, it is clear that $(n-1)!$ possible Hamiltonian cycles exist. Note that counting the number of Hamiltonian cycles in this case is akin to determining the number of circular permutations of 4 items. This is because various different standard permutations (node orderings) are actually equivalent cycles that are simply shifted (e.g. $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ and $1 \rightarrow 2 \rightarrow 3 \rightarrow 0$).

As the phase estimation algorithm tests each eigenstate individually, it was necessary to list all 6 eigenstates via the constant `EIGENSTATES` in `solver.py`. Note that the number of orderings to test for a symmetric network is actually 3 as noted by [4], as opposite but equal paths have the same length; however, testing all 6 states explicitly allows my implementation to handle asymmetric networks as well

5 Results and Discussion

I was able to successfully implement [4]'s algorithm on fully-connected networks of 4 nodes. Specifically, my implementation used 6 estimator qubits just like [4]; I found that increasing this number during testing by even 1 qubit greatly slowed down the calculation speed. Given that [4] only test their solution on one toy example 4-node network, it is impossible to compare my implementation's results with theirs. Instead, I evaluated the performance of my algorithm on 4 randomly-generated sample graphs which might be more representative of real-world TSP problems. The adjacency matrix of one such sample is below in Figure 2. The specifics for each test graph are hosted on my GitHub in the `data` folder.

$$\begin{bmatrix} 0 & 0.6542 & 0.1557 & 1.0521 \\ 0.6542 & 0 & 1.5267 & 1.4883 \\ 0.1557 & 1.5267 & 0 & 0.3388 \\ 1.0521 & 1.4883 & 0.3388 & 0 \end{bmatrix}$$

Figure 2: The adjacency matrix for `graph_0.txt`, one of the test networks.

Although this is inherently the simplest case of TSP problems, the algorithm remains imperfect, likely because of an insufficient number of estimator qubits. Even [4] note that their implementation's expected results differ from the actual sampled results. I applied the algorithm to each sample graph 5 times, counting the number of instances where it selected the correct route (having the minimum eigenvalue estimate). My results are below. Furthermore, running the algorithm using an additional estimator qubit did indeed up performance, especially on test graph 1 on which the solver increased its accuracy by 40%. This came at a substantial increase in compute time however.

6 Future Work

Substantial future work remains regarding quantum approaches to TSP, both with respect to this QPE approach and other quantum optimization schemes. Some brief, non-comprehensive thoughts regarding QPE specifically are below.

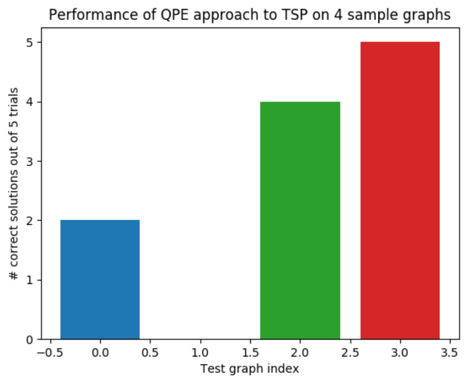


Figure 3: Number of successful trials out of 5 for each randomly-generated sample graph.

6.1 Implementing with a noise model

At the start of the project, I had hoped to test the efficacy of my algorithm under a standard noise model such as the one from pyQuil's `add_decoherence_noise`. However, `add_decoherence_noise` only operates on QUIL programs using the Rigetti QVM's native gate set of `RZ, RX`, and `CNOT`. Despite attempts at re-compiling my program into the native gate set (testing different device frameworks and timeout settings), I was unable to do so due to timeouts by the quilc compiler. I suspect this was due to the complexity of re-compiling my custom gates coupled with the low compute available to me. Consequently, there remains an opportunity for future work to characterize the extent to which noise affects the QPE results.

6.2 Experimenting with more qubits

My computer struggled to run experiments on the qvm with even 16 qubits. Given that the effectiveness of the QPE approach is bounded by the number of estimator qubits used, further work to characterize the effects of more qubits would be very interesting.

7 Acknowledgements

Thank you to the CS269Q instuctors and course staff for a wonderful course! The instructors are welcome to post this paper and share the source code as they see fit.

References

[1] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[2] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[3] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[4] Karthik Srinivasan, Saipriya Satyajit, Bikash K Behera, and Prasanta K Panigrahi. Efficient quantum algorithm for solving travelling salesman problem: An ibm quantum experience. *arXiv preprint arXiv:1805.10928*, 2018.

[5] Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial optimizationeureka, you shrink!*, pages 185–207. Springer, 2003.