Solving the Traveling Salesman Problem Using QAOA

Daniel Henry
dhenry@stanford.edu

Sofia Josefina Lago Dudas
sdudas@stanford.edu

June 4, 2019

---

1 Using the Codebase

Our public repository can be found at `https://github.com/danielhenry1/QAOA_TSP`. Our implementation of the Quantum Approximate Optimization Algorithm can be found in the file 'qaoa.py.' To run this code, a user will need the following packages in a Python virtual environment - numpy, scipy, pyquil and the Rigetti QVM. The QVM server will need to be started and running. The user then simply needs to run the Python file and our program will execute the QAOA algorithm and provide an optimal solution on a sample graph we have provided in the file. If the user wants to adjust the provided adjacency matrix, the number of cities, or the value of p in the QAOA, these values are clearly defined as globals at the top of the file.

2 Project Overview

We implemented the Traveling Salesman Problem (TSP) using the Quantum Approximate Optimization Algorithm (QAOA) on Rigetti's QVM. The TSP is an optimization problem that minimizes the distance of a path visiting all cities in the given set of cities. The TSP and its variants have many applications in fields ranging from microchip manufacturing to DNA sequencing. The Traveling Salesman Problem is in the class of NP-hard problems, meaning that any classical solution will run in superpolynomial time. The naive solution to this problem, that checks all possible routes, runs in time $O(n!)$. We have included such a naive solution to the TSP in our codebase in the file `tsp.py`. However, this problem is susceptible to a quantum speedup, and implementation on a quantum computer can be used to find the solution more efficiently.

In this project, we used the QAOA to generate solutions for the TSP when n is small, specifically getting QAOA to give optimal results when $n = 3$ and distances between the cities both symmetric, and asymmetric. Although the algorithm running on the QVM is slower than implementations on classical hardware today due to repeated calls to the quantum virtual machine server, we were able to generate a mapping from the problem to a Hamiltonian that has a ground state with a solution to the TSP. This proof of concept shows that it is possible to

use quantum computers in a unique way to solve the TSP that is theoretically more efficient, although in practice slower.

3 Implementation

We modeled our solutions using the same format as Homework Project 2 Problem 3, as follows. Let $x_{\alpha,j}$ be a set of n^2 binary variables where α denotes a city and j denotes a time step in some path. Thus $x_{2,0}$ =1 means the path starts in city 2 in the zeroth time step. Likewise $x_{4,n-1} = 1$ means the path end in city 4 on the last time step. There are n time steps as we will visit each city exactly once. We form a bitstring that represents a path as follows:

$$x_{0,0}x_{1,0}, ..., x_{n1,0}x_{0,1}, ..., x_{n-1,n-1}$$

In our implementation, we attempted to express the hamiltonian constraints as Pauli Sum operators that we could apply to the wave function and calculate the expectation of. We tried to create parallel constraints to the homework, but had to make a few modifications. For example, we created a Pauli Sum to represent the Hamiltonian that penalizes paths such that a city that appear more than once, or not at all, in a given path. Additionally, we created a Pauli Sum to represent the Hamiltonian that penalizes timesteps with more than one city in it. However, instead of explicitly ensuring that we could only go between cities where an edge exists, we assumed the graph was fully connected and represented by an adjacency matrix where every pair of cities had a double representing the distance. In the use case where no edge exists between a city, we would advise setting this entry of the adjacency matrix to `FLOAT_MAX` or an equivalent.

The QAOA begins in a superposition over all possible solutions to the problem. The total number of qubits used in the TSP is n^2, and so the total number of possible solutions is 2^{n^2}. We begin by applying a Hadamard gate to all qubits, generating this superposition over all possible states.

In order to solve the problem, we had to generate a Cost Hamiltonian with a ground state that is a solution to the TSP. To do this, at the beginning of the program we had to build a Hamiltonian that penalized (1) states that repeated locations more than once, (2) states that had multiple locations in a single timestep, and (3) states that had long distances between stops. We think of (1) and (2) as constraints as they are non-negotiables in solutions, and (3) as a penalty, as long distances between stops makes a solution inefficient. We created Pauli Sum terms to penalize all three of these penalties, however weighing (1) and (2) more heavily so that an invalid solution would never be chosen over an inefficient one. We modeled our Hamiltonian using Pauli Sums that would penalize states that were invalid or inefficient solutions. We were able to unify the implementation of (1) and (2) constraints with a function in our code called `penalize_range()`.

What we realized was shared about penalties (1) and (2) was that both required that the only states over a certain range of qubits be of the form 0^i10^{n-i-1} for some $i < n$. For (1) this was in a certain range of a timestep, as there should only be one city visited, and for (2) this was in a range of all of the timesteps corresponding to a certain city, as each city should

only be visited once. Thus, we found the appropriate ranges to penalize, and passed into a the function `penalize_range()` that penalized all states not comprised of all 0s with one 1.

To penalize a range of qubits $\{q_1, q_2, ...q_n\}$, we quickly realized that the Pauli expression $Z(q_1) \otimes Z(q_2) \otimes ... \otimes Z(q_n)$ penalized all bitstrings of $\{q_1, q_2, ...q_n\}$ that were of the form $0^i 10^{n-i-1}$ for some $i < n$, except the state 1^n. Thus, we needed an additional term that penalized just the state 1^n.

We realized that the matrix representation of $I - Z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$, and thus $I - Z$ tensored with itself n times would look, in matrix form, as $\begin{bmatrix} 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ ... & & & \\ 0 & 0 & ... & 2 \end{bmatrix}$, which we could weight with a constant to penalize just the $\{1, 1, ..., 1\}$. Thus, our constraint penalties were a linear combination of $Z(q_1) \otimes Z(q_2) \otimes ... \otimes Z(q_n) + (I(q_1) - Z(q_1)) \otimes (I(q_2) - Z(q_2) \otimes (I(q_3) - Z(q_3) \otimes ... \otimes (I(q_n) - Z(q_n))))$

After generating the Cost Hamiltonian, we also built a Driver Hamiltonian. This driver Hamiltonian is parameterized in addition to the Cost Hamiltonian, and is included to ensure that the space over the states is changing, allowing for the superposition to move and ultimately settle into a ground state. This Driver Hamiltonian is defined as $I(q_0) - \sum_{i=0}^{n^2} X(q_i)$

These Hamiltonians, the Cost and the Driver, are exponentiated and parameterized using two angle-dependent operators. We then take an expectation over these parameterized states and classically optimize the angles used in these operators to discover the ground state of the Hamiltonians. We take a final expectation once the optimization has converged to find the likely ground states the algorithm has discovered.

4 Results

We were extremely pleased with our results for $n = 3$. Initially, we didn't have an effective cost penalty to penalize the state 111. We got the following result, in which there are two valid solutions represented as top results, but less likely than the inaccurate initial result. The majority of the invalid states were of the form 111 over the range, meaning that our method of applying penalties was working correctly, we just had yet to penalize the 111 state.

After we began penalizing the 111 state for specified ranges in the Cost Hamiltonian, we immediately saw results that put valid solutions as the most likely and removed the solutions that included the 111 state for most timesteps. See Figure 2. At this point, we didn't change our method of penalizing terms any further. At this point, we were weighting equally constraints (1) and (2) (which penalized invalid states), with (3) (which penalized long paths).

We realized that the weight penalizing long paths was comparable to the weights penalizing invalid solutions, which instead should be non-negotiables. We then increased the weight that

would penalize invalid solutions, particularly all those that were not of the form 111 as those were more frequently the form of invalid solutions. We saw that we were able to get all 6 valid solutions in our top 10 most likely states. See Figure 3.

Finally, by further adjusting the relative weights between our two constraints and by setting $p = 3$, we achieved a result in which all 6 valid states were the most likely states with a notable drop to the next most likely state, the optimal result from the algorithm. See Figure 4.

5 Future Extension

There are a number of directions that the project could go in from this point. First, we would like to experiment with a more generalized formula for effective weights to give to the penalties as n increases. However, we were unable to generate much analysis on this front or on cases where n is large due to runtime. Even when n was increased to 4, we saw an enormous increase in runtime, which is expected as the number of possible states in the wavefunction grows at 2^{n^2} and quickly becomes unwieldy on the QVM. We were also interested in doing work on finding analytic ways to find optimal angles γ and β which would lead to enormous speedups in the program. We would also be interested in better defining what the runtime of the QAOA would be, and whether it would be possible to show quantum advantage using this algorithm or others like it for large n. One result that was surprising to us was that we did not see significant improvements to our results with an increase in p. We achieved similarly successful results for p values between 3 and 10. We expected to have much learer results for the higher p values. Perhaps it is simply because our results for $p = 3$ were already good, but we are interested in exploring in more detail how changes in p can impact the results.

We are comfortable with this report in its entirety as well as all of our codebase being published online on the CS269Q course website. Thanks for a wonderful class!

6 Works Consulted

1. Edward Farhi, Jeffrey Goldstone, Sam Gutmann. A Quantum Approximate Optimization Algorithm. 2014. URL https://arxiv.org/abs/1411.4028

2. Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G. Rieffel, Davide Venturelli, Rupak Biswas. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. 2017. https://arxiv.org/abs/1709.03489

3. Lou K. Grover. A fast quantum mechanical algorithm for database search. 1996. URL https://arxiv.org/abs/quant-ph/9605043

4. Karthik Srinivasan, Saipriya Satyajit, Bikash K. Behera, Prasanta K. Panigrahi. Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience. 2018. URL https://arxiv.org/abs/1805.10928

5. Dominic J. Moylett, Noah Linden, Ashley Montanaro. Quantum speedup of the Travelling Salesman Problem for bounded-degree graphs. URL https://arxiv.org/abs/1612.06203

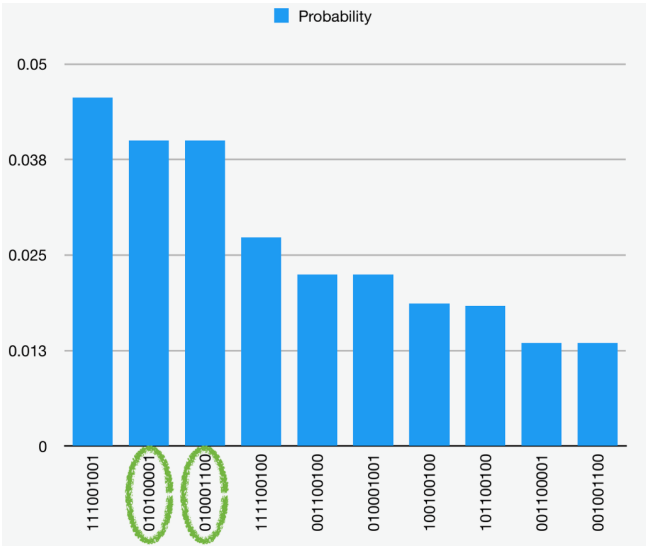

Figure 1: Ineffective Cost Hamiltonian

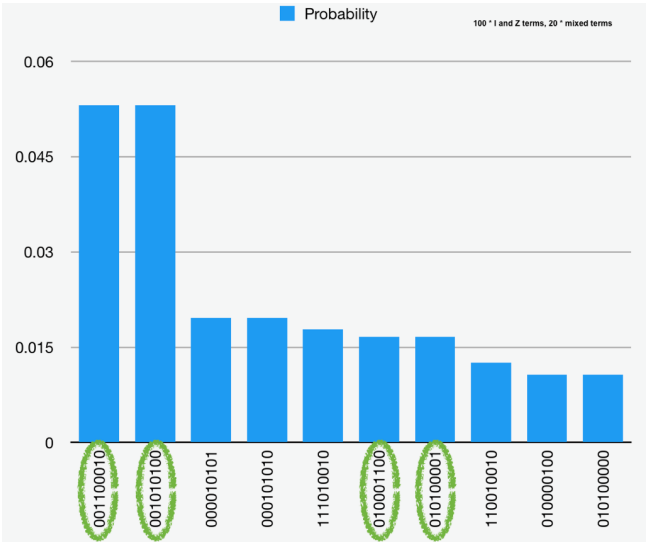

Figure 2: Improved Cost Hamiltonian

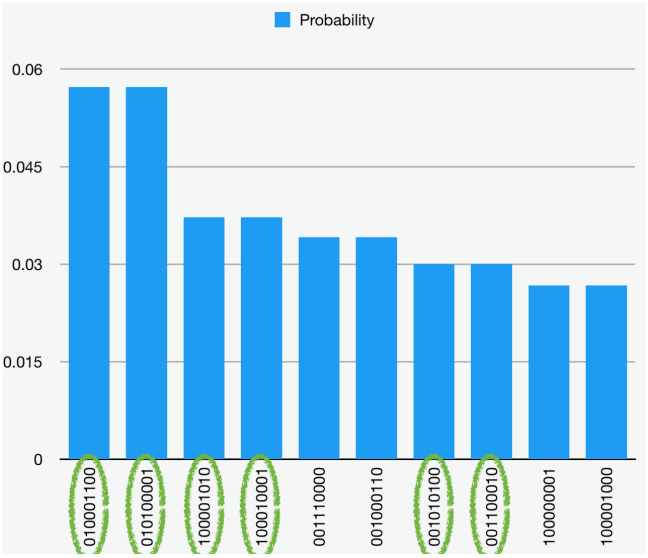

Figure 3: Adjusted Weights Leads to All Valid States Likely

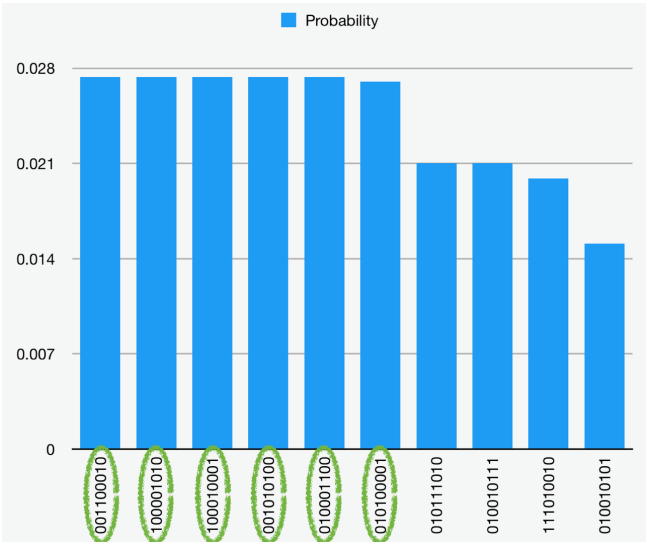

Figure 4: Further Tuning Created Optimal Result