A QAOA solution to the traveling salesman problem using *pyQuil*

Matthew Radzihovsky, Joey Murphy, Mason Swofford

May 2019

1 Problem statement

The traveling salesman problem (TSP) is a famous NP-complete optimization task. In the TSP we are presented a graph of cities with weighted edges that represent the distance between the cities. We are then asked to find the shortest path in the graph that visits all of the cities exactly once. TSP is a classic combinatorial-optimization problem which, in the worst case, must enumerate every possible path and check its value. However, there are many classical heuristics which can do much better than this O(n!) runtime. Classical algorithms which rely on solving relaxed problems and then polishing for feasible solutions, such as the branch-and-bound algorithm, often achieve far superior performance from the expected runtime of the naive approach [1]. However, in the worst case, these are still O(n!). Given how closely TSP relates to other combininatorial-optimization problems and its NP-completeness, any method which improves upon the factorial run time would revolutionize the world.

Recently, quantum algorithms have offered novel solutions to this canonical problem. Examples include solutions via phase estimation [2] and a Quantum Approximate Optimization Algorithm (QAOA) [3]. Expanding on our Project 2 problem, we implement a QAOA [4] solution in *pyQuil* [5]. To do so, we defined a Hamiltonian that enforces the constraints of the problem; namely a Hamiltonian that penalizes paths that visit cities more than once, paths that go between cities without edges, and paths that are long. Therefore, by finding the ground state of our Hamiltonian, we can efficiently find a solution to this NP-hard problem.

We build on this solution by implementing so-called mixer operators, as described in [3]. TSP is an optimization problem where we are interested in determining a solution within a specified subspace. Within this subspace there are hard constraints that must always hold, such as the cities must have a path connecting them. In addition, there are soft constraints, which refer to constraints we want to optimize over, such as the total path length. Therefore, as Hadfield *et al.* describe, it is possible to refocus on the unitary operators rather than Hamiltonians to more efficiently implement mixers.

For all our methods, we formulate the problem as finding the bitstring which corresponds to the optimal sequence of cities. Using n^2 binary variables, where n is the number of cities, if we divide the bitstring into n continuous sections of n variables, each section should have exactly one variable with value 1 and the rest should be 0. The variable that is 1 corresponds, according to its index within its section, to the city that we have chosen to visit during that section, which maps directly to a timestep. To be precise, if our bitstring is x, then $x_{ni+j} = 1$ with $0 \leq i, j, \leq n-1$, corresponds to visiting the jth city in the ith timestep.

2 Methods

QAOA uses a few general steps: 1. Prepare the initial state, 2. Apply the cost Hamiltonian, 3. Apply the driver Hamiltonian, 4. Exponentiate and parameterize the resulting product in p steps by p γ's and p β's, and 5. Use classical algorithms to optimize over β and γ. It is important at each minimization step to store the best bitstring sampled so far in order to have the best path in the end. Within the general algorithm for QAOA, there are several design choices to be made, such as the specific choices of cost and driver Hamiltonians as well as the γ and β values. Additionally, since we are dealing with both soft and hard constraints, we must develop a weighting scheme. Each of these choices may also introduce hyperparameters to the problem for which values might be tuned by hand or included in the classical optimization step.

2.1 Standard QAOA

The first theoretical task of this project is to select our driver and cost Hamiltonians. For our driver Hamiltonian, we choose a simple expression that has a known ground state, namely:

$$\hat{\mathcal{H}_D} = \sum_{i \in G} X_i \tag{1}$$

which has ground state:

$$|H_D^0\rangle = \frac{1}{2^{|G|}} \sum_{z \in \{0,1\}^{|G|}} |z\rangle \tag{2}$$

For our cost Hamiltonian, building off Project 2, we have four terms that each penalize a constraint of the TSP problem. Such a Hamiltonian makes use of binary constraints to increase the costs for paths which violate the constraints.

1. Each city should be visited exactly once, which corresponds to the Hamiltonian term:

$$\hat{\mathcal{H}_1} = \sum_{\alpha=0}^{n-1} \big(1 - \sum_{j=0}^{n-1} x_{\alpha,j}\big)^2 \tag{3}$$

where n is the number of cities to visit and $x_{\alpha,j} \in \{0,1\}$ is a binary variable indicating that we visit city α along some path x at time step j.

2. Each timestep should only contain one city (i.e. we can't be in multiple cities at once), corresponding to the term:

$$\hat{\mathcal{H}_2} = \sum_{j=0}^{n-1} \big(1 - \sum_{\alpha=0}^{n-1} x_{\alpha,j}\big)^2 \tag{4}$$

where the variables are defined identically to above, however, now we sum over timesteps j at the end.

3. An edge from one city to another should only be taken if the two cities are connected, corresponding to the term:

$$\hat{\mathcal{H}_3} = \sum_{j=0}^{n-2} \Big(\sum_{\alpha=\{x_{\alpha,j}=1\}} \sum_{\beta=\{x_{\beta,j+1}=1\}} \xi \mathbb{1}_{\{E_(\alpha,\beta) \notin G\}} \Big) \tag{5}$$

where $\xi \in \mathbb{R}$ is a tunable hyperparameter scaling the penalty factor and

$$\mathbb{1}_{\{E_(\alpha,\beta) \notin G\}} = \begin{cases} 1 & \text{if edge does not exist} \\ 0 & \text{if edge exists.} \end{cases}$$

4. Each city should be visited exactly once, which is given by the Hamiltonian term:

$$\hat{\mathcal{H}_4} = \sum_{j=0}^{n-2} \Big(\sum_{\alpha=\{x_{\alpha,j}=1\}} \sum_{\beta=\{x_{\beta,j+1}=1\}} \lambda_{\alpha,\beta} \Big) \tag{6}$$

where $\lambda_{\alpha,\beta}$ represents the distance traveled from α to β if the edge exists. If the edge between α and β does not exist, then we use the largest distance in the graph to penalize the path. Mathematically we define it as:

$$\lambda_{\alpha,\beta} = \begin{cases} W(\alpha,\beta), & \\ \text{if edge exists} & \\ & \\ \max(W(l,m)), & \\ \forall E(l,m) \in G & \text{if edge does not exists} \end{cases}$$

where W is the distance matrix.

We construct our cost and driver Hamiltonians in *pyQuil* by representing them as sums of Pauli matrices. This allows us to efficiently implement the time-evolution by using the `pyquil.paulis` module.

2.2 QAOA using Mixers

Additionally, we implement TSP QAOA using mixers as discussed in [3] and adapted from code orignally written by Michal Stechly[1]. As described in Hadfield et al., using partial mixers which explore paths within the hard constraints, we can create a new driver Hamiltonian based on these mixers. There are value-selective ordering swap mixing Hamiltonians, which swap cities visited at time i and j in a path if they are specific cities denoted u and v, and value-independent order swap mixing Hamiltonians, which swap cities regardless of cities visited at i and j. Mathematically, we can represent value-selective ordering swap mixing Hamiltonian for cities $\{u,v\}$ at time $\{i,j\}$ as:

$$\hat{\mathcal{H}}_{\{i,j\},\{u,v\}} = \tag{7}$$

$$\sum_{l:\{i,j\}=\{u,v\}} |(i_1,\ldots,l_{i-1},v,\ldots,l_{j-1},u,\ldots,l_n\rangle \tag{8}$$

$$\langle(i_1,\ldots,l_{i-1},v,\ldots,l_{j-1},u,\ldots,l_n| \,. \tag{9}$$

We can also represent the value-independent order swap mixing Hamiltonians by summing over the $\binom{n}{2}$ value-selective swaps as:

[1]Find his original Github repository here.

$$\hat{\mathcal{H}}_{\{i,j\}} = \sum_{\{u,v\}\in\binom{n}{2}} \hat{\mathcal{H}}_{\{i,j\},\{u,v\}} \tag{10}$$

When encoding such a mixer, they simplify into a combination of Pauli Gates implemented as follows [3]:

$$\hat{\mathcal{H}}^{(enc)}_{\{i,j\},\{u,v\}} = S^+_{u,i}S^+_{v,j}S^-_{u,j}S^-_{v,i} + S^-_{u,i}S^-_{v,j}S^+_{u,j}S^+_{v,i} \tag{11}$$

where

$$S^+ = X + iY = |1\rangle\langle 0| \tag{12}$$

$$S^- = X - iY = |0\rangle\langle 1| \tag{13}$$

Therefore, we can define our parameterized driver unitary using our definition for $\hat{\mathcal{H}}^{(enc)}_{\{i,j\},\{u,v\}}$ above as follows:

$$\mathcal{U}_{D:\{i,j\},\{u,v\}}(\beta) = e^{-i\beta H_{\{i,j\},\{u,v\}}}. \tag{14}$$

This parameterized driver Hamiltonian differs from the standard case, because instead of choosing a Hamiltonian with a known ground state, we use mixers to allow our driver to explore solutions within the hard constraints of TSP. We explore the performance of both of these implementations more in the following section.

3 Results

In order to test our quantum algorithms, we built a pipeline that generates random connected weighted graphs, with varying average node degrees, and then solve these graphs using a brute force classical algorithm. We can now guarantee we have found the optimal path, and can therefore sanity check our quantum implementations against it. While this will be infeasible for large graphs, because the quantum alogrithms are run on simulators, they are much slower and the classical algorithm is not currently a bottleneck.

As for our quantum algorithms, our mixer implementation is successful at solving the TSP and reproduces, up to cyclic permutations, the solution found by our classical solver. In testing our mixer implementation, we find that with 500 samples of the output, with Trotterization expansion order set to 2, and a graph containing 3 cities, the execution time is about 10 seconds. Reducing the Trotterization expansion to first order, the program still returns the correct solution and executes in about 5 seconds. We find that when increasing the number of cities to greater than or equal to four, the execution time is prohibitively long ($\sim$ 10 minutes) if the Trotterization expansion is not held to a first order approximation. With 500 output samples, Trotterization expansion order equal to 1, and 4 cities in the graph, the mixer implementation finds the solution in about 5 minutes. These tests were conducted using a full-connected graph.

Our TSP QAOA implementation using cost Hamiltonians is successful at finding the classical solution for graphs with cities up to 4, but requires at least about 1000 circuit output samples for consistent results. The cost Hamiltonian solution is slightly faster than the mixer implementation, requiring only about 10 seconds on a 2015 Macbook Pro to solve the TSP for 3 cities with 1000 samples and default Trotterization expansion order of 2. For graphs larger than 4, the cost Hamiltonian implementation's execution time is prohibitively long.

4 Discussion

Between the two implementations, properties such as the number of gates required may contribute to differences in execution time for graphs of the same size. Namely, the number of gates in our cost Hamiltonian solution scales as n^2, where n is the number of cities in the graph. On the other hand, each of the two terms in Equation 11 can be implemented as a sum of 8 terms, with each term being the product of four Pauli matrices [3]. The mixer in Equation 14 can then be implemented using $(n-1)\binom{n}{2}$ of these 4-qubit gates, implementable then in a depth $2\kappa \leq 2n$ gates, where $\kappa = n$ for even n and $\kappa = n - 1$ for odd n. [3] §5.1.2 contains a more detailed discussion of the gate requirements for the mixer implementation. In short, the mixer approach requires less gates than the cost Hamiltonian implementation. However, as noted in §3, from our testing we find the cost Hamiltonian solver is generally faster than the mixer. This may be due to various other confounding factors, such as calls to external libraries in the mixer implementation, or a larger number of parameters in the classical optimization step.

Both implementations require n^2 qubits. In future work, we hope to explore solutions to the TSP that use qubits wisely, beating the n^2 requirement.

5 Conclusion and Future Work

We have investigated and implemented two methods of QAOA for TSP. It is exciting that we are able to solve an NP-hard problem such as TSP by using quantum algorithm and classical minimization. We hope to continue exploring better implementations of the TSP and quantum algorithms for other NP-hard problems. This implementation provides a glimpse into what problems quantum computers can solve and the possibility of utilizing quantum supremacy.

In theory, quantum algorithms have been proven to surpass classical algorithms. However, in implementing our QAOA algorithm for TSP, we find there is always the challenge of taking measurements to get information from a quantum system that must be done with smart choices in order to maintain an advantage. This does not even take into account the increased errors currently with quantum bits, that can lead to incorrect solutions or an increased need for error correction. These challenges lead to many exciting possibilities for quantum algorithms to be developed even before we have reliable many qubit quantum computers.

Due to the prohibitive time constraint of running larger quantum systems using a simulator, we were not able to experiment with how our quantum algorithms compared against classical algorithms on real world scale problems. However, this would be an exciting area of future research, especially as quantum hardware continues to improve and this becomes feasible.

6 Code

Our code and installation instructions are publicly available at https://github.com/murphyjm/cs269q_radzihovsky_murphy_swofford.

References

[1] T. Volgenant and R. Jonker, "A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation," *European Journal of Operational Research*, vol. 9, no. 1, pp. 83 – 89, 1982.

[2] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi, "Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience," *arXiv e-prints*, p. arXiv:1805.10928, May 2018.

[3] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz," *arXiv e-prints*, p. arXiv:1709.03489, Sep 2017.

[4] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv e-prints*, p. arXiv:1411.4028, Nov 2014.

[5] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," 2016.