Quantum Computing: Finding a Quantum Analog to a Classical Algorithm

Andrew Rippy

Department of Physics, Wabash College, Crawfordsville, IN 47933

(Dated: June 24, 2024)

Abstract

This paper will endeavor to show a quantum analog to the traveling salesman problem, an NP problem which proves to be exceedingly difficult to solve in reasonable time classically. Can we even utilize quantum computing to solve a NP-Hard problem in a more efficient manner? The answer is yes, we can, and this paper will describe a circuit that can do so.

I. INTRODUCTION

The travelling salesman problem (TSP) is as follows, "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?" This problem is considered NP-hard in terms of optimization, which is important in theoretical computer science.

In the theory of computing, as mentioned prior belongs to the class of NP-complete problems. This means that the worst-case running time for any algorithm for the TSP increases superpolynomially (with an upper bound of exponentially) with the number of cities.

The classical decision version of the TSP (where given a length L, the task is to decide whether the graph has a tour of at most L). The problem[1] was first proposed in 1930 and is one of the most intensively studied problems in optimization and computation theory, and thus is used as a benchmark for many optimization methods regarding NP-problems. Even though the problem is computationally difficult, many exact algorithms are known, so that some instances with thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction, the only drawback being significantly long computation times, as it grows superpolynomially, hence the desire for a quantum method to take advantage of the parallelism and significant run time improvements.

II. THE TRAVELING SALESMAN PROBLEM: A CLASSICAL APPROACH

A direct solution is to move through permutations of possible cycles and see which one is cheapest, the brute force method. However, as if you are familiar with permutations, the run time for this brute force approach grows with a polynomial factor of $\mathcal{O}(n!)$, so this solution quickly becomes impractical, as even for something as small as only 15 cities has

possibly over 1 trillion cycles to test. With that option clearly out of the question, we move to a more optimized algorithm.

The Held–Karp algorithm is a dynamic programming algorithm proposed in 1962.[1] It is an exact solution to this problem, and other equivalent algorithms, such as the Hamiltonian cycle problem, in exponential time. The exact run time of this algorithm is $\mathcal{O}(n^2 2^n)$.

A. An explanation of the Held-Karp algorithm

First begin by numbering the cities 1 to n, and designate an arbitrary starting city to be called 1. Note that since we are looking for an optimal *cycle*, it does not matter which city we begin with. The algorithm begins by calculating for each set of cities, $\{S \mid S \subseteq \{2, \cdots, n\}\}$ and $\{e \in S^c/\{1\}\}$, the shortest 1 way path from 1 to e that passes through every city contained in S, but not through any other cities contained in S^c. Denote this distance traveled as $g(S, e)$, and denote the distance between two adjacent cities u, v as $d(u, v)$. For example, when $S = \{\emptyset\}$, then calculating $g(S, e)$ is just $d(1, e)$. Similarly, consider $g(\{2\}, 3)$ or $g(\{2, 3\}, 4)$, which would just be the length of the path $1 \to 2 \to 3$, or the shortest choice between $1 \to 2 \to 3 \to 4$ and $1 \to 3 \to 2 \to 4$ respectively. Rather than computing all the possible paths, we can take advantage of known sub-paths to cut down on computation time. For example, if the path $1 \to 2 \to 3 \to 4$ is longer than $1 \to 3 \to 2 \to 4$, and thus the path $1 \to 2 \to 3 \to 4 \to 5$ is known to be longer than $1 \to 3 \to 2 \to 4 \to 5$, so we need not calculate that path's distance. Thus, as we calculate the shortest possible path of each subset of cities, calculating paths which contain that subset becomes faster.

Generalizing this further, suppose $S = \{s_1, \cdots, s_k\}$, and $S_i = S/\{s_i\}$. If the shortest path from 1 through S to e has s_i as its second to last city, then removing the final edge from this path must give the shortest path from 1 to s_i through S_i. Thus there are only at most k possible shortest paths from 1 to e through S that one must calculate, with length $g(S_i, s_i) + d(s_i, e)$, and $g(S, e) = \min_{\leq i \leq k} g(S_i, s_i) + d(s_i, e)$. Once we have completed this for every S, we now have the shortest path distance distance from 1 to any other city i. The final part of the algorithm takes these distances and adds $d(i, 1)$ to give $n - 1$ possible cycles to compute, and thus the shortest cycle then becomes the minimum of these possible cycles.

III. EXPLANATION OF THE QUANTUM ALGORITHM PROPOSED

A. A quick explanation of a city distance matrix

We can express the distance between any two cities in a compact and useful manner by using a distance matrix, where the distance between city i and city j is the value indicated in the a_{ij} entry of the matrix. For example, in the case of 3 cities, this is represented by a 3×3 distance matrix.

$$a = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 9 & 17 \\ 15 & 9 & 0 & 8 \\ 20 & 17 & 8 & 0 \end{bmatrix}. \tag{1}$$

The a_{ii} entry is represented by a zero, since the distance from a city to itself is 0. The distances are also two-way, so it is the same distance from city i to city j as city j to i, giving us a symmetric matrix. Reading the above matrix, the distance from city 1 to city 2 would be 10, city 1 to city 3 would be 15, and city 2 to city 3 would be 9. This notation is important, as I will soon describe.

B. The quantum distance matrix

By inspection, the aforementioned distance matrix alone is not unitary in general. Thus, we would need to make a few adjustments to convert this to a quantum setting.[2] We want to be able to make the distances add as we compute paths to test for the shortest. For this, we know that when we multiply phases of a quantum state, they add together in the coefficients. Thus, if we encode the distances from each city into the phase of the coefficient, it will add as expected and not affect the magnitude. Therefore, we define the quantum distance matrix

$$A \equiv \begin{bmatrix} e^{i\phi_{1\to1}} & e^{i\phi_{1\to2}} & e^{i\phi_{1\to3}} & e^{i\phi_{1\to4}} \\ e^{i\phi_{2\to1}} & e^{i\phi_{2\to2}} & e^{i\phi_{2\to3}} & e^{i\phi_{2\to4}} \\ e^{i\phi_{3\to1}} & e^{i\phi_{3\to2}} & e^{i\phi_{3\to3}} & e^{i\phi_{3\to1}} \\ e^{i\phi_{4\to1}} & e^{i\phi_{4\to2}} & e^{i\phi_{4\to3}} & e^{i\phi_{4\to4}} \end{bmatrix}. \tag{2}$$

We are not done yet, as we now need to convert A into a unitary matrix. To do this, we will split it into several parts that encode the information of A, into one unitary matrix per

city in the network. For example, city 1 would be expressed as:

$$e^{i\phi_{1\to1}}\left|00\right\rangle\left\langle 00\right| + e^{i\phi_{2\to1}}\left|01\right\rangle\left\langle 01\right| + e^{i\phi_{3\to1}}\left|10\right\rangle\left\langle 10\right| + e^{i\phi_{4\to1}}\left|11\right\rangle\left\langle 11\right|, \tag{3}$$

or, equivalently written in matrix form:

$$U_1 \equiv \begin{bmatrix} e^{i\phi_{1\to1}} & 0 & 0 & 0 \\ 0 & e^{i\phi_{2\to1}} & 0 & 0 \\ 0 & 0 & e^{i\phi_{3\to1}} & 0 \\ 0 & 0 & 0 & e^{i\phi_{4\to1}} \end{bmatrix}. \tag{4}$$

In general, the distance matrix for each city will be of the form:

$$U_j \equiv \begin{bmatrix} e^{ia} & 0 & 0 & 0 \\ 0 & e^{ib} & 0 & 0 \\ 0 & 0 & e^{ic} & 0 \\ 0 & 0 & 0 & e^{id} \end{bmatrix}, \tag{5}$$

where a, b, c, d represent the distances encoded for each respective city pairing. To construct each of these matrices, we decompose them into:

$$U_j = \Big(\begin{bmatrix} 1 & 0 \\ 0 & e^{i(c-a)} \end{bmatrix} \otimes \begin{bmatrix} e^{ia} & 0 \\ 0 & e^{ib} \end{bmatrix} \Big) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i(d-c+a-b)} \end{bmatrix} = \begin{bmatrix} e^{ia} & 0 & 0 & 0 \\ 0 & e^{ib} & 0 & 0 \\ 0 & 0 & e^{ic} & 0 \\ 0 & 0 & 0 & e^{id} \end{bmatrix}. \tag{6}$$

The two 2×2 matrices can be expressed by the phase shift gates $\big(P(c-a)\big)$ and $\big(P(b-a), P(a)\big)$, which is just the identity and a phase shifted entry in the lower right index. Then for the full final unitary matrix, we would take the tensor product of all the unitary operators:

$$U = U_1 \otimes U_2 \otimes U_3 \otimes U_4. \tag{7}$$

Finally, in order for this to work properly, we need to convert this into a controlled operator, one that only applies the operation provided the control qubit is $|1\rangle$, similar to a C-NOT gate. Thus we have:

$$CU = CU_1 \otimes CU_2 \otimes CU_3 \otimes CU_4. \tag{8}$$

For each CU_i, each component gate would become a control gate. That is, $P(c-a)$ becomes $CP(c-a)$, $CP(b-a)$. Further, we need to make the full U_j itself controlled, creating a

controlled controlled U_j matrix. For the 4 city example, this then becomes an 8×8 control matrix for each CU_j, giving us:

$$CU_j \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{ia} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{ib} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & e^{ic} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{id} \end{bmatrix}. \tag{9}$$

To encode the distance into the phase, we simply normalize by 2π to bound the distance between 0 and 2π once we know the full range of distances. We can use the quantum phase estimation algorithm to estimate the eigenvalues of U. Note that given the way we have constructed U, it is a diagonal matrix since it is the tensor product of n diagonal matrices, which means the eigenstates of U are the computational basis states with eigenvalues of the corresponding diagonal elements.

C. Encoding the cycles

In formal terms, a full route for the salesman is called a Hamiltonian cycle.[2] Because this is a complete graph, there are a total of $n!$ possible Hamiltonian cycles in a graph with n nodes. In the example case of 4 cities that we used before, there would be $4! = 24$ possible Hamiltonian cycles. However, that does not mean we necessarily need to check all of them, this is because we only need to consider *distinct* cycles. For example, any shift of a cycle will be the same cost, that is, the following cycles would be equivalent:

$1 \to 2 \to 3 \to 4$
$2 \to 3 \to 4 \to 1$
$3 \to 4 \to 1 \to 2$
$4 \to 1 \to 2 \to 3$

Thus we only need consider one from each distinct cycle shift equivalence class. It is

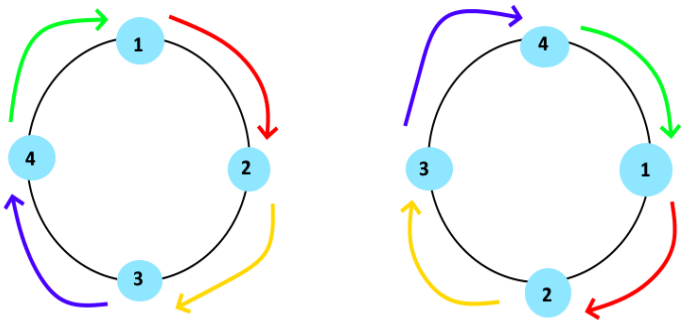

FIG. 1. An example of a $1 \to 2 \to 3 \to 4$ cycle shift

important to note that the paper from which I drew this information from argued that there are only 3 distinct cost classes, but this either makes further assumptions that were not clearly stated, or is just blatantly wrong. I chose to omit this and stick with the 6 equivalence classes I know are correct for certain. In the instance of 4 cities, there are a total of 6 such equivalence classes:

$1 \to 2 \to 3 \to 4$
$1 \to 2 \to 4 \to 3$
$1 \to 4 \to 2 \to 3$
$1 \to 4 \to 3 \to 2$
$1 \to 3 \to 2 \to 4$
$1 \to 3 \to 4 \to 2$

Using cycles shifts on these 6 distinct cycles will generate all of the 24 possible Hamiltonian cycles. Sparing the combinatorial proof, we know that the number of distinct cycles given n cities is given by $\frac{(n-1)!}{2}$, provided the cost from traveling to city $i \to j$ being the same as $j \to i$. If this is not the case, then there are at most $(n-1)!$. Now we can generate a distinct eigenstate for each equivalence class of cycles. The state

$$|\psi\rangle = \otimes_j |i(j) - 1\rangle \text{ where } j \in \{1, 2, \cdots, n\}. \tag{10}$$

For the example cycle $1 \to 2 \to 3 \to 4$, $i(1) = 4$, since this means from city 4 we traveled to city 1. Now that we have a way of denoting the cycle, we convert it to binary.

$$|i(1) - 1\rangle = |4 - 1\rangle = |3\rangle = |11\rangle \,. \tag{11}$$

Thus, the 4 total states that would encode this information would be:

$$|i(1) - 1\rangle = |4 - 1\rangle = |11\rangle \,, \tag{12}$$
$$|i(2) - 1\rangle = |1 - 1\rangle = |00\rangle \,, \tag{13}$$
$$|i(3) - 1\rangle = |2 - 1\rangle = |01\rangle \,, \tag{14}$$
$$|i(4) - 1\rangle = |3 - 1\rangle = |10\rangle \,. \tag{15}$$

Now we combine these into a tensor product to describe the cycle:

$$|11\rangle \otimes |00\rangle \otimes |01\rangle \otimes |10\rangle = |11000110\rangle \,. \tag{16}$$

For the cycle $1 \to 2 \to 4 \to 3$, we apply the same process and get:

$$|i(1) - 1\rangle = |3 - 1\rangle = |10\rangle \,, \tag{17}$$
$$|i(2) - 1\rangle = |1 - 1\rangle = |00\rangle \,, \tag{18}$$
$$|i(3) - 1\rangle = |2 - 1\rangle = |01\rangle \,, \tag{19}$$
$$|i(4) - 1\rangle = |4 - 1\rangle = |11\rangle \,. \tag{20}$$

Here is the binary conversion for each of the equivalence classes:

$1 \to 2 \to 3 \to 4$	$\|11000110\rangle$
$1 \to 2 \to 4 \to 3$	$\|10000111\rangle$
$1 \to 4 \to 2 \to 3$	$\|10001101\rangle$
$1 \to 4 \to 3 \to 2$	$\|01001110\rangle$
$1 \to 3 \to 2 \to 4$	$\|11001001\rangle$
$1 \to 3 \to 4 \to 2$	$\|01001011\rangle$

IV. PUTTING IT TOGETHER

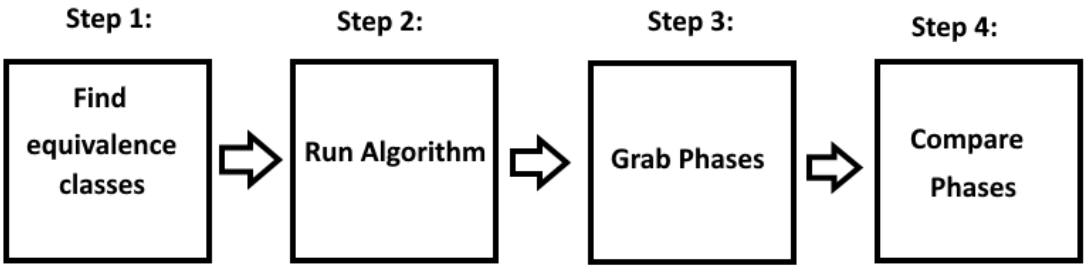

FIG. 2. Finding the shortest path

Now that we have all the components we need, we can describe how the circuit would work. Note that the eigenstates we must measure directly correspond to the equivalence classes of the cycles. Thus, we only need run this computation for each cycle equivalence class, or eigenstate. For each of these distinct equivalence classes, we run them through the four CU_i gates, then use the Inverse QFT to rip off the resulting phase after moving through the cycle. We then compare the distance for each equivalence class, and find our shortest path.

V. FUTURE WORK AND CONCLUSION

Though the traveling salesman problem is difficult to solve in a timely manner classically, utilizing the benefits of quantum parallelism and quick circuit timing. It only grows based

on the time required to compute per each equivalence class, still giving $\mathcal{O}(n!)$ run time as n gets large, but taking advantage of the speed of quantum parallelism, this growth is mitigated, and unavoidable for a full analysis of the equivalence classes. Note that this is just one method of applying Quantum Computing to a classical problem. There could be other options towards solving the problem in a different manner.

---

[1] R. Bellman, *Dynamic programming treatment of the travelling salesman problem*, (Journal of Assoc. Computing, 1962)

[2] K. Srinivasan, S. Satyajit, B. Behera, and P. Panigrahi. *Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience.* (Qiskit, 2018.)