A Quantum Algorithm for the Bottleneck Travelling Salesman Problem

by

Raveel Tejani

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

in

The Faculty of Science

(Physics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2024

Abstract

chapter not complete.

Table of Contents

List of Tables

List of Figures

Acknowledgements

chapter not complete.

Chapter 1

Introduction

Continuous advancements in quantum computing have opened new ways to solve complex computational problems that were previously considered intractable using classical methods. Well known examples of these advancements include Shor's [1] and Grover's [2] algorithm for factoring and unstructured search respectively. The Bottleneck Traveling Salesman Problem (BTSP) serves as a challenging optimization problem in the field of logistics and operations research. Its practical applications range from optimizing vehicle routes to circuit design. Consequently, achieving an efficient solution is highly sought after. However, the BTSP is classified as an NP-hard problem, indicating it is at least as difficult as the hardest problems in the NP class. For reference, an NP problem must satisfy two conditions: no known solution in polynomial time, and a solution can be verified in polynomial time. An NP-hard problem does not need to satisfy the verification condition. To address this problem, many heuristic approaches are explored [3][4]. However, inherent to such methods is a trade-off between precision and efficiency.

chapter not complete.

Chapter 2

Theory

2.1 The Bottleneck Travelling Salesman Problem

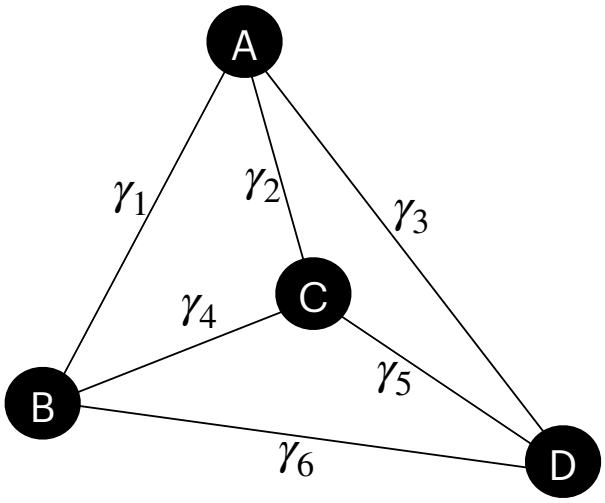


Figure 2.1: An undirected weighted graph represention of a symmetric 4-city system. The vertices represent cities and the edge weights represent the cost of travel.

The BTSP can be represented as a graph problem. We start with a graph, whose vertices are labelled A through D, representing a 4-city system. We define movement from one vertex to another as a walk, done through the edges connecting our vertices. We are interested in a particular walk known as a Hamiltonian cycle that contains every vertex exactly once before returning to the start. Our graph also includes edge weights, we define as γ_i. The BTSP is to find the Hamiltonian cycle in a graph, where the largest edge weight (bottleneck) is minimized. This is distinct from the Travelling Salesman Problem (TSP) where the combined edge weights in a given cycle is minimized. The total possible Hamiltonian cycles is given by $(N-1)!$, where N is the number of nodes. We present a symmetric case in FIG .2.1, $N_k \to N_{k+1} = N_{k+1} \to N_k = \gamma_i$. Thus the total possible cycles is $(N-1)!/2$. It is important to note that a solution to either BTSP or TSP is not unique. BTSP solutions also do not necessarily equate to the TSP solutions. We can illustrate an example below. Consider all the Hamiltonian cycles for a symmetric 4-city system:

$$A \to B \to C \to D \to A$$
$$A \to B \to D \to C \to A$$
$$A \to C \to B \to D \to A$$

Assigning some arbitrary weights, we can see the total costs of the cycles below. The first cycle is the solution to BTSP as its largest edge weight at 5 is the smallest among all three. The last cycle is a solution to the TSP as its combined edge weight is the smallest.

$$\gamma_1 + \gamma_4 + \gamma_5 + \gamma_3 = 4 + 4 + 5 + 4 = 17$$
$$\gamma_1 + \gamma_6 + \gamma_5 + \gamma_2 = 4 + 6 + 5 + 2 = 17$$
$$\gamma_2 + \gamma_4 + \gamma_6 + \gamma_3 = 2 + 4 + 6 + 4 = 16$$

By simply changing the weight of γ_6 to 5, we can illustrate all cycles are solutions to the BTSP.

$$\gamma_1 + \gamma_4 + \gamma_5 + \gamma_3 = 4 + 4 + 5 + 4 = 17$$
$$\gamma_1 + \gamma_6 + \gamma_5 + \gamma_2 = 4 + 5 + 5 + 2 = 16$$
$$\gamma_2 + \gamma_4 + \gamma_6 + \gamma_3 = 2 + 4 + 5 + 4 = 15$$

The computational complexity of the BTSP is known to be NP-hard. Implying there is no algorithm for a solution in polynomial time. A brute-force approach would imply that we can run an algorithm in $O((N-1)!)$ time.

2.2 Quantum Computing Basics

2.2.1 Frequently Used Notation

Single qubit states:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

2-qubit states:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$|00\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, |01\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Multiple qubit states are achieved through tensor products of single states. An n-qubit system will span a Hilbert space $N = 2^n$. We can see with the 2-qubit system we have 4 states. Quantum logic gates serve as the quantum computing counterparts to classical logic gates, performing operations on qubits instead of classical bits. Represented as unitary matrices, these operations must be reversible. Below we can see the Hadamard gate and the result of its application to single qubit states, frequently employed to establish a uniform superposition.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

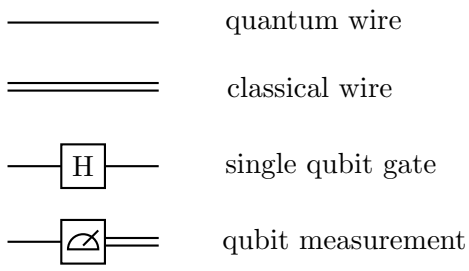
$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$$
$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$$

We can refer to equation 1 to see the result of Hadamard on an n-qubit system. Let's have a look at the Hadamard gate applied to $|00\rangle$:

$$\begin{aligned} H^{\otimes 2}|00\rangle &= H|0\rangle H|0\rangle \\ &= |+\rangle|+\rangle \\ &= (\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle))(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) \\ &= \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle + |1\rangle) \\ &= \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \end{aligned}$$

Here we can see why Dirac notation is powerful as it simplifies tensor products and removes need for matrix representation.

2.2.2 Quantum Circuit Components



2.2.3 Controlled Gates and Phase Kickback

section not complete

1. We will describe the use of control gates.
2. We will describe the emergence of phase kickback with the use of hadamard gates discussed in the previous section. This will be important to know going into the next section.

2.3 Quantum Phase Estimation

The phase estimation algorithm initially proposed by Alexey Kitaev [6] plays an important role as a subroutine for the more widely known factoring algorithm by Peter Shor [1]. We first must briefly discuss the Quantum Fourier Transform (QFT) as it is key to understanding phase estimation [5]. Given a computational basis state $|x\rangle$, applying the QFT (F_N) results in:

$$F_N|x\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i x k N^{-1}} |k\rangle$$

Let's represent this in binary notation and decompose it into a tensor product. We can represent $|x\rangle$ as a string of bits, and the QFT as a tensor product of single qubit basis states:

$$|x\rangle = |x_1 x_2 ... x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes ... \otimes |x_n\rangle$$

$$F_N|x\rangle = \frac{1}{\sqrt{2^n}} \bigotimes_{j=1}^{n} (|0\rangle + e^{2\pi i x 2^{-j}}|1\rangle) \tag{2.1}$$

$$= \frac{1}{\sqrt{2^n}} ((|0\rangle + \omega_1|1\rangle) \otimes (|0\rangle + \omega_2|1\rangle) \otimes ... \otimes (|0\rangle + \omega_n|1\rangle))$$

$$\begin{aligned} \omega_1 &= e^{2\pi i x 2^{-1}} = e^{2\pi i (0.x_n)} \\ \omega_2 &= e^{2\pi i x 2^{-2}} = e^{2\pi i (0.x_{n-1} x_n)} \\ &... \\ \omega_n &= e^{2\pi i x 2^{-n}} = e^{2\pi i (0.x_1 ... x_n)} \end{aligned}$$

An important characteristic of the w_j is the bit shift occurring in the exponent. If we look at w_1, the exponent has a factor $x2^{-1}$, which is equivalent to one right bit shift: $x_1 ... x_{n-1}.x_n$. Integer multiples of the exponent would imply full rotations returning to the same point thus we can ignore all the values on the left of the decimal and what remains is $0.x_n$.

Let's discuss the phase estimation problem. Given an eigenstate $|\lambda\rangle$ of a unitary operator U, we want to calculate a good approximation for $\phi \in [0, 1)$ satisfying:

$$U|\lambda\rangle = e^{2\pi i \phi}|\lambda\rangle \tag{2.2}$$

The phase estimation algorithm uses two registers of qubits. The first one will be a set of n control qubits that determine the precision of our approximation. The second register will be a set of m qubits initialized to an eigenstate $|\lambda\rangle$.

Let's walk through the quantum circuit in FIG. 2.2, to understand the inner workings of this algorithm. Our initialized state is $|0^{\otimes n}\lambda\rangle$. From here we perform the same operation we find in equation 1, where all the qubits in the first register are set to a uniform superposition on all states 2^n. The next portion of the algorithm involves applying controlled gates based on the unitary operator U. The function of these CU gates is to apply the operator U on $|\lambda\rangle$ if the control qubit is in the state $|1\rangle$. We can have a look at the effect on the n^{th} qubit, after it has been prepared in a superposition by the Hadamard gate:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |\lambda\rangle = |0\lambda\rangle + |1\lambda\rangle$$

Applying CU and factoring out the eigenstate:

$$\begin{aligned} CU\frac{1}{\sqrt{2}}(|0\lambda\rangle + |1\lambda\rangle) &= \frac{1}{\sqrt{2}}(CU|0\lambda\rangle + CU|1\lambda\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\lambda\rangle + e^{2\pi i\phi}|1\lambda\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\phi}|1\rangle) \otimes |\lambda\rangle \end{aligned}$$

We can see the eigenstate after the CU operation is left unchanged. the phase has been encoded into the control qubit instead, a result is due to the phase kickback. Thus, we can

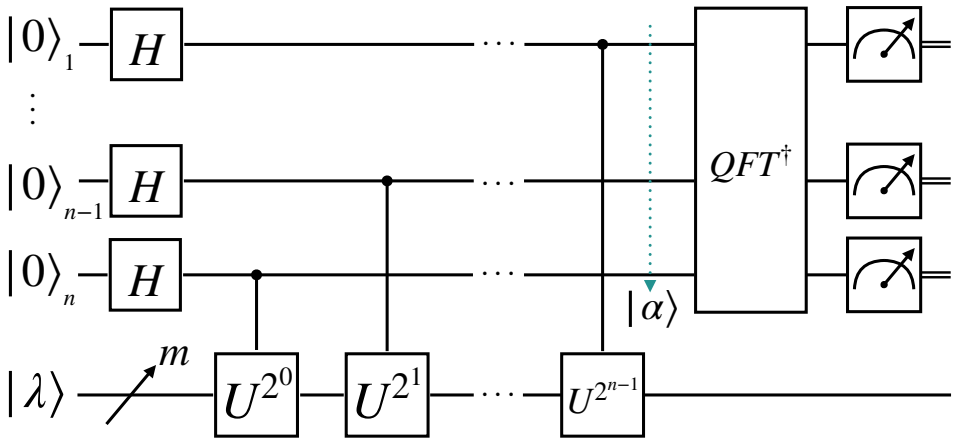


Figure 2.2: A quantum circuit representation of the phase estimation algorithm. Given, $U|\lambda\rangle = e^{2\pi i\phi}|\lambda\rangle$, this algorithm allows us to generate an approximation for $\phi \in [0, 1)$. The circuit consists of two registers of qubits, the first n-qubits are initialized to $|0\rangle$ and contribute to the precision of the ϕ value obtained. The second register of m-qubits is initialized to the eigenstate of U. The Hadamard gates, H, are used to create a uniform superposition in the first register. The control gates based on U are responsible for encoding phase to the qubits in the first register. Finally, a $QFT^\dagger$ is performed on the first register to extract the encoded phase. Each subsequent qubit in the first register would require double the control gates. Thus, with a large n we obtain a more precise value for ϕ, but also exponentially increase our computation time.

reuse our eigenstate for the next qubit. Consecutive qubits have double the amount of CU operators as the previous, thus squaring the eigenvalue each time:

$$\begin{aligned} \text{qubit}_{\text{n-1}} &: |0\rangle + e^{2\pi i\phi 2}|1\rangle \\ &... \\ \text{qubit}_1 &: |0\rangle + e^{2\pi i\phi 2^n}|1\rangle \end{aligned}$$

We know that the value of $\phi < 1$. We can represent this in binary notation in the form $0.\phi_1\phi_2...\phi_n$:

$$\phi = \sum_{j=1}^{n} \phi_j 2^{-j}$$

If we have another look at the control qubits using binary notation for ϕ instead, we can see the result of the multiple CU operations simply results in right bit shifts:

$$\begin{aligned} \text{qubit}_{\text{n}} &: |0\rangle + e^{2\pi i(0.\phi_1\phi_2...\phi_n)}|1\rangle \\ \text{qubit}_{\text{n-1}} &: |0\rangle + e^{2\pi i(0.\phi_2...\phi_n)}|1\rangle \\ &... \\ \text{qubit}_1 &: |0\rangle + e^{2\pi i(0.\phi_n)}|1\rangle \end{aligned}$$

If we look at the form of first register after all the CU operations in the state $|\alpha\rangle$, it will resemble the result of performing the QFT we saw in equation 5. Where our ω_j are:

$$\begin{aligned}
\omega_1 &= e^{2\pi i x 2^{-1}} = e^{2\pi i(0.\phi_n)} \\
\omega_2 &= e^{2\pi i x 2^{-2}} = e^{2\pi i(0.\phi_{n-1}\phi_n)} \\
&\ldots \\
\omega_n &= e^{2\pi i x 2^{-n}} = e^{2\pi i(0.\phi_1 \ldots \phi_n)}
\end{aligned}$$

simply performing the inverse QFT will give us $|\phi\rangle = |\phi_1\phi_2...\phi_n\rangle$. We can immediately see the approximation is limited by the number of qubits in the first register. A simple strategy would be to increase the number of qubits; however this would also increase computational cost as we double our use of CU gates for each additional qubit.

Chapter 3

Algorithm for the Constraint Problem

The constrained version of the BTSP asks whether there exists a Hamiltonian cycle in which the weight of every edge is less than a specified threshold α. In such a cycle, if we denote the weight of any edge as γ_i, then it must satisfy the condition:

$$\gamma_i < \alpha$$

We will construct the algorithm in the following steps:

1. Normalize edge weights so no single hamiltonian cycle is greater than or equal to 1.

2. Construct a unitary operator that holds information regarding the hamiltonian cyles as phases in the diagonal.

3. Set all edgeweights $> \alpha$ to zero and construct a secondary unitary operator similiar to step 2.

4. Create controlled gates using the unitary operators constructed in Step 2 and 3.

5. Identify all the eigenstates of the unitary operators that map to the phases associated with the hamiltonian cycles.

6. Perform phase estimation twice with an eigenstate using the control gates to evaluate the hamiltonian cycle before and after the edgeweights $> \alpha$ are set to zero.

7. Compare the two phases achieved. If they are equal, the corresponding hamiltonian cycle is a solution that satisfies the constraint.

3.1 Normalize Edge Weights

A hamiltonian cycle of a complete graph with N nodes requires N edge-weights to complete the cycle. A single graph consists a total of $N(N-1)$ edgeweights in a directed graph. we can choose the largest N and use these to normalize the edge weights. Let w describe our edge-weights and will be a list of $m = N(N-1)$ elements:

$$w = \{w_1, w_2, \ldots, w_m\}$$

Sort w in descending order to obtain:

$$w' = w'_1, w'_2, \ldots, w'_m$$

Where: $w'_1 \geq w'_2 \geq \ldots \geq w'_m$

The sum S of the largest N items in w can be described as:

$$S = \sum_{i=1}^{N} w'_i$$

We can now perform the normalization. Let $\tilde{w}$ describe our normalized edge-weights:

$$\tilde{w} = (S + \epsilon)^{-1} w$$

Where: $\epsilon > 0$. The purpose of ϵ is to make sure if any normalized hamiltonian cycle is exactly equal to S then we do not have a zero phase.

3.2 Unitary Operator for Hamiltonian Cycles

We start by constructing diagonal matrices U_j, one for each node in a complete graph and describe the matrix elements:

$$[U_j]_{kk} = e^{2\pi i \gamma_{jk}(1-\delta_{jk})} \tag{3.1}$$

Where: $1 \leq j, k \leq N$ N denotes the total number of nodes. γ_{jk} represents the edgeweight connecting node $j \to k$

Then we construct U, a tensor product of all the diagonal matrices:

$$U = \bigotimes_{j}^{N} U_j$$

U will be a $N^N \times N^N$ matrix with only the diagonal elements populated. Because the diagonal elements will entirely consist of phases $[U]_{kk} = e^{i\alpha_{kk}}$. We can confirm the unitary operator condition is satisfied: $U^\dagger U = \mathbb{1}$.

3.3 Identifying the Eigenstates associated with the Hamiltonian Cycles

Given U's diagonal nature, its eigenstates align with the basis vectors. Our focus is on specific eigenstates corresponding to the Hamiltonian cycles, determined by the phases. To comprehend how the diagonal elements of U derive from the individual U_j matrices, we visualize the tensor product construction. Notably, the product populates the diagonal elements of U allowing a simplification where we consider these elements directly:

$$
\begin{aligned}
[U]_0 &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_{N-1}]_0 \cdot [U_N]_0 \\
[U]_1 &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_{N-1}]_0 \cdot [U_N]_1 \\
&\vdots \\
[U]_{N-1} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_{N-1}]_0 \cdot [U_N]_{N-1} \\
[U]_N &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_{N-1}]_1 \cdot [U_N]_0 \\
&\vdots \\
[U]_{2N-1} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_{N-1}]_1 \cdot [U_N]_{N-1} \\
[U]_{2N} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_{N-1}]_2 \cdot [U_N]_0 \\
&\vdots \\
[U]_{N^2-1} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_0 \cdot [U_0]_{N-1} \cdot [U_N]_{N-1} \\
[U]_{N^2} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_1 \cdot [U_{N-1}]_0 \cdot [U_N]_0 \\
&\vdots \\
[U]_{N^2+N-1} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_1 \cdot [U_{N-1}]_0 \cdot [U_N]_{N-1} \\
[U]_{N^2+N} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_1 \cdot [U_{N-1}]_1 \cdot [U_N]_0 \\
&\vdots \\
[U]_{N^3-1} &= [U_1]_0 \cdot [U_2]_0 \cdot \ldots \cdot [U_{N-2}]_{N-1} \cdot [U_{N-1}]_{N-1} \cdot [U_N]_{N-1}
\end{aligned}
$$

This pattern indicates:

$$
[U]_k = [U_1]_{\alpha_{N-1}} \cdot [U_2]_{\alpha_{N-2}} \cdot \ldots \cdot [U_{N-2}]_{\alpha_2} \cdot [U_2]_{\alpha_1} \cdot [U_N]_{\alpha_0} \tag{3.2}
$$

With:

$$
\alpha_i = \left(k // \left(N^i\right)\right) \% N
$$

Where:
// denotes integer division
% is the modulus operation

We can see the pattern follows base-N counting where N is the total number of nodes. Simply converting k into base-N, will give us the indices of our original diagonal matrices. To locate the relevent eigenstates, we start by identifying the elements in U_j associated with a hamiltonian cycle, retrieve their respective indices, convert this string of indices from Base-N to k. and we would have identified our eigenstate $|k\rangle$. We will see a concrete example in the following section

Chapter 4

Constraint Solution for a 4-City Problem

4.1 Algorithm Construction

for our contstraint test, we simply need a tensor product of 4 matrices:

$$U = U_A \otimes U_B \otimes U_C \otimes U_D$$

Using equation 3.1, we can construct the diagonal elements of our marices. Let $|u_i\rangle = \text{diag}(U_i)$:

$$|u_A\rangle = \begin{bmatrix} 1 \\ e^{i\gamma_{AB}} \\ e^{i\gamma_{AC}} \\ e^{i\gamma_{AD}} \end{bmatrix} |u_B\rangle = \begin{bmatrix} e^{i\gamma_{BA}} \\ 1 \\ e^{i\gamma_{BC}} \\ e^{i\gamma_{BD}} \end{bmatrix} |u_C\rangle = \begin{bmatrix} e^{i\gamma_{CA}} \\ e^{i\gamma_{CB}} \\ e^{i\gamma_{BC}} \\ e^{i\gamma_{BD}} \end{bmatrix} |u_D\rangle = \begin{bmatrix} e^{i\gamma_{DA}} \\ e^{i\gamma_{DB}} \\ e^{i\gamma_{DC}} \\ 1 \end{bmatrix}$$

Refering to equation 3.2 regarding the matrix elements for U, the diagonal elements for the 4-city problem will reduce to:

$$[U]_k = [U_A]_{\alpha_3} \cdot [U_B]_{\alpha_2} \cdot [U_C]_{\alpha_1} \cdot [U_D]_{\alpha_0} \tag{4.1}$$

With:

$$\alpha_i = \left(k // \left(4^i\right)\right) \%4$$

lets construct the full tabe:

1. In this section I will work through the steps for algorithm construction mentioned in the previous chapter for the 4-city problem.
2. a circuit figure will be included as well.

4.2 Results: Simulations with Qiskit

section not complete.

progress report: The algorithm has been constructed and tested with simulators in Qiskit for the 4-city problem. The full solution has been explored but given time constrained may not be solved and will be briefly discussed in the following section. The simulator does generate graphs we intend to include in this section. the presentation style of the graphs is still being explored. A constraint problem

Hamiltonian Cycle Details		
Cycle	**Edge Weights Sum**	**Matrix Elements Product**
$A \to B \to C \to D \to A$	$\gamma_{AB} + \gamma_{BC} + \gamma_{CD} + \gamma_{DA}$	$a_1 \cdot b_2 \cdot c_3 \cdot d_0$
$A \to B \to D \to C \to A$	$\gamma_{AB} + \gamma_{BD} + \gamma_{DC} + \gamma_{CA}$	$a_1 \cdot b_3 \cdot d_2 \cdot c_0$
$A \to C \to B \to D \to A$	$\gamma_{AC} + \gamma_{CB} + \gamma_{BD} + \gamma_{DA}$	$a_2 \cdot c_1 \cdot b_3 \cdot d_0$
$A \to C \to D \to B \to A$	$\gamma_{AC} + \gamma_{CD} + \gamma_{DB} + \gamma_{BA}$	$a_2 \cdot c_3 \cdot d_1 \cdot b_0$
$A \to D \to B \to C \to A$	$\gamma_{AD} + \gamma_{DB} + \gamma_{BC} + \gamma_{CA}$	$a_3 \cdot d_1 \cdot b_2 \cdot c_0$
$A \to D \to C \to B \to A$	$\gamma_{AD} + \gamma_{DC} + \gamma_{CB} + \gamma_{BA}$	$a_3 \cdot d_2 \cdot c_1 \cdot b_0$
Conversions		
Rearranged Indices (Base 4)	**Base 10**	**Base 2**
1230	108	01101100
1302	114	01110010
2310	180	10110100
2031	141	10001101
3120	216	11011000
3201	225	11100001

Table 4.1: Comprehensive Hamiltonian Cycles Analysis

simulation with 5-cities might also be explored, again if time permitting. Otherwise things are fairly on track.

next steps:

1. **complete the sections presented in this thesis, particularly regarding theory and algorithm construction section**
2. **complete this chapter of the constraint solution with circuit figures and graphs from Qiskit Simulation**
3. **complete other sections (discussion, intro, abstract...) including appendix regarding the code used in python.**
4. **time permitting: construct and present 5-city simulation as well**

Chapter 5

Discussion

1. this section will dicuss the advantages and drawbacks of the algorithm proposed.

Bibliography

[1] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[2] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[3] John Larusic. A heuristic for solving the bottleneck traveling salesman problem. 2005.

[4] Ravi Ramakrishnan, Prabha Sharma, and Abraham P. Punnen. An efficient heuristic algorithm for the bottleneck traveling salesman problem. *Opsearch*, 46(3):275–288, 09 2009. Copyright - Operational Research Society of India 2009; Last updated - 2023-08-22.

[5] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[6] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995.

Appendix A

Code for the 4-city constraint problem

1. this section will include any code used to produce graphs in the simulated section.