Implementing quantum travelling salesman for IBM Qiskit Aqua

Allison Newman

Department of Computing

East Tennessee State University

Johnson City, TN, USA

newmanal@etsu.edu

Abstract–We propose implementing a quantum algorithm to solve the Travelling Salesman Problem (TSP) for the IBM Qiskit Aqua Library. The TSP, which solves for the shortest complete circuit between all points on a fully-connected graph, is applicable to many fields of applied science research as well as practical commercial interests including routing logistics, genome sequencing, and CNC machine automation. Belonging to the complexity class of NP, precise solutions to the TSP are classically intractible; however, leveraging the inherent computational speedup of quantum computers can provide a more efficient means of finding solutions. The IBM Qiskit Aqua library is an open-source project that aims to provide quantum algorithms to researchers looking to explore the power of quantum computers. Our code and documentation will be provided in the form of a Jupyter Notebook and made available for use within the Aqua library.

1. INTRODUCTION

Quantum computing is a new field of computer science that seeks to introduce a new paradigm for computing faster than classical computers. Research into algorithms that are uniquely efficient on quantum computers has been ongoing since the introduction of Shor’s algorithm for integer factorization in 1994 [1]. In particular, the intractable algorithms in NP are desperate for a more efficient solution that could be provided by a quantum computer – so much so that the P = NP problem is currently the most important question in theoretical computer science [2].

One such NP problem to be examined is the Travelling Salesman Problem. Precise solutions of the TSP can be calculated in no better than O(n22n) [3], becoming intractable at impractically small values of n; even the best approximation algorithms are limited to values of n in the thousands – requiring potentionally hundreds of CPU days to complete [4]. Thus is it critical to explore the capabilities of a quantum algorithm in solving the TSP. Such algorithms can currently be simulated using the Qiskit Aqua library [5], an open source project that provides applied science researchers the ability to utilize these quantum algorithms to advance their work. It is the intent of this project to contribute to the Qiskit library with the code and documentation of a TSP algorithm based on the algorithm proposed by Srinivasan et al [6].

P vs NP

*Definition 1* – **P (Polynomial Time)** The set of problems whose solutions can be deterministically computed by an algorithm with polynomial time complexity [7].

*Definition 2* – **NP (Non-deterministic Polynomial Time)** The set of problems whose solutions can be verified in polynomial, but whose solutions cannot be found deterministically by an algorithm of polynomial time complexity. Such problems require algorithms of exponential time to solve deterministically, or otherwise rely on a non-deterministic algorithm to approximate a solution [7].

A longstanding question in theoretical computer science, and one of seven Millenium Prize Problems [8], P vs NP asks for proof of set equivalence between P and NP. That is, can a determinstic polynomial time algorithm be found to solve all problems currently in NP? Most computer scientitists believe that P ≠ NP [9], prompting the need for new, faster ways to computing such problems.

The Travelling Salesman

The Travelling Saleman Problem (TSP) is one such NP problem. The TSP supposes a travelling salesman wishes to begin at his home city, visit some number of other cities, and then return home, visiting every city exactly one time and doing so in the shortest distance possible. The most simple form of this problem dictates that all cities are connected to one another, with some associated distance to travel between each, and that the distances are symmetric – travelling from city A to city B is the same distance as travelling from city B to city A. TSP is represented as a fully-connected, symmetrically weighted graph (V,E), where and , where n is the number of cities in the graph (including the starting city).

The brute-force search method of solving the TSP requires searching every possible route (tour) the salesman could take to find the shortest tour. In a graph of n-cities, there exists tours. This can be reduced to given symmetic weights, as half of the tours are merely the reverse of other tours. One can see that the limiting factor in such an algorithm is the rapid growth of the factorial function. Consider a TSP of n = 20, thus

 

tours exists that need to be checked. If one were able to check the distance of 3.0 × 109 tours per second (a reasonable clock speed for a consumer-grade CPU), it would take over 38 years to find the shortest tour.

Requiring decades for such calculations is unacceptable for the many practical applications of the TSP. The most obvious of uses applications of TSP exist in routing problems. Every school district in the United States faces the problem of forming optimal bus routes to pick up students. A typical school bus with maximum occupancy of 72 students poses a TSP of potentially n = 72: find the shortest route to retrieve all students from their homes and then return to the school. UPS delivers an average of 15 million packages per day with each truck making an average of 120 deliveries. This then represents thousands of TSPs of n = 120. Even at these relatively small datasets, our a brute-force approach reaches runtimes that exceed the age of the universe. Consider that some applications such as routing the drilling sequence of a CNC router, mapping a telescopes trajectory to photograph stars in the universe, or searching through a genome sequence can involve potentially thousands of points of navigation [4], we see that advancements in the TSP algorithms must be made, or some problems would remain forever unsolvable.

In 1962, Held and Karp presented a Dynamic Programming approach for solving TSP in O(n22n). Thusfar no algorithm exists to find an exact solution for TSP in faster than O(2n). Approximation methods for solving TSP have been devised that can find a near-solution much faster, however. Most notably is the Concorde TSP Solver written by Applegate *et al.* [4]. In 2006, Concorde found a solution at n = 85900, requiring 192 CPU years of computation time on a cluster of 250 processors.

Quantum Computing

First theorized by Feynman in 1981 [10], the possibility of a quantum computer for years represented the possibility to use incredible power of quantum mechanics to store and manipulate information. Four years later, Deutsch outlined the Quantum Turing Machine, a reimagining of a classical Turing Machine allowing for the unique properties of quantum mechanics, such as the inclusion of probability amplitudes [11], [12]. This new paradigm also brings new restrictions: quantum algorithms must be logically reversible – no information can be lost in the process of moving between states [13].

The zeal for realizing a quantum computer, however, was not spurred until the introduction of Shor’s quantum algorithm for integer factorization [1]. This new algorithm not only had dire practical applications – threatening the security of RSA encryption – but did so in polynomial time, an exponential speedup over the best classical algorithms.

The rate of growth within classical computing hardware is also projected to slowdown. Moore’s Law – a prediction that the size of integrated circuit shrinks by half every 18 to 24 months, thus increasing computing power on a chip – has seen a decline in accuracy over the past decade [14]. It is expected that this trend of slower improvement in classical circuitry will continue into the future. Quantum computers then provide an alternative for faster computing, at a time when classical computing innovation is slowing to a crawl.

Quantum computing relies on a qubit’s (quantum bit) property of superposition, wherein it exists in some combination of the 0 and 1 states. When the qubit is measured, however, it is read as either a 0 or 1 discretely. The chance of a qubit being measured as 0 or 1 can be described by probability amplitudes – equal to the squared absolute value of the coefficient associated with each state [14]. More information on the notation describing these qubit systems is provided in Appendix A.

There are two approaches to be examined when building a quantum computer and the algorithms to run on them, quantum annealing (adiabatic) [15] and universal gate-model quantum computers. Annealing quantum computers rely on near-zero (Kelvin) temperatures to function, but can offer potentially thousands of qubits. While the annealing model offers more qubits, these macines are more limited in the algorithms they can perform; they are unable, in particular, of performing Shor’s or Grover’s algorithms [16]. The universal-gate model relies on subjecting the qubits to a series of unitary transformations, and is more consistent with classical programming techniques. As such, his project focuses on universal gate-model quantum algorithms and Noisy Intermediate-Scale Quantum (NISQ) computing.

The NISQ paradigm, as the name suggests, involves quantum computers that are both noisy and intermediate in scale. One primary roadblock towards advancing quantum computing is the increased level of “noise” or errors that is introduced to a quantum system as more qubits are added. Quantum systems are also notoriously difficult (impossible) to measure without causing some sort of change in the system [17]. As more qubits or more gates are added to a system, the probability that errors occur in the final measurement is increased. The effects of these errors are shown by Amico, Saleem, and Kumph whose experiments with Shor’s Algorithm on IBM Q failed to successfully factor at N=35 due to the increased number of gates required [18].

IBM offers quantum cloud computing through their IBM Q Experience and Qiskit frameworks. Currently they have four machines available throughout the world, with the most advanced machine running 20 qubits [19].

The quantum TSP algorithm we propose to explore is the Srinivasan et al. algorithm from *Efficient quantum algorithm for solving travelling salesman problem: An IBM Quantum Experience (2018)* [6]. This publication details the necessary steps for completing the algorithm on a quantum computer. In particular, the quantum gate circuit (Figure 1) is given, displaying the steps for calculating the length of a single to

The algorithm at n=4 uses fourteen qubits, all initialized to the state. Eight qubits are passed through Pauli-X gates (shown in red) such that the state of this 8-qubit register forms an state representing the index of the given tour, in this case . The remaining six qubits are passed through Hadamard gates. Those six qubits then act as the control-qubits for unitary transformations on the tour register, before being passed through the quantum fourier transform and then measured. The measured data is relayed through classical means into a database of tour lengths. This process is repeated for every tour in the graph. Once complete, Grover’s quantum search algorithm [20] can be used to find the minimum tour length. See Appendix B for more information on the quantum gates used here.

Software Tools

Qiskit is an open-source framework created by IBM for the development and proliferation of quantum algorithms. The framework includes four levels of depth at which researchers and programmers can access depending on their needs or skillsets (Figure 3). The framework is aimed at near-term quantum algorithms, associated with the current NISQ paradigm of quantum computing. The Terra level is the lowest level of development in the framework, concerning itself with the circuitry of the device [21]. Aer provides a simulator in C++ focused on running the circuits previously designed in Terra [22]. Ignis is for developers concerned with quantum error correction (QEC) and reducing the noise of quantum circuits [23]. Finally, Aqua allows programmers to add higherlevel code, written in Python, for quantum algorithms to the Qiskit library [24]. These algorithms can then be used by researchers in science and business.

The Jupyter Project is an organization that aims to provide an interactive programming experience through their Jupyter Notebooks [25]. Juptyter Notebooks are setup as programmable cells, wherein the cells can execute code with many available languages. The cells also support HTML markup for formatting and media, and LaTeX for mathematical documentation [26]. Jupyter Notebooks are editted and executed within a browser, allowing for easy sharing and collaboration on a given project.

A screenshot of a cell phone

Description automatically generatedPurpose

*Fig 1. Circuit Diagram from Srinivasan et al. (2018)*

This project aims to contribute the code for a quantum algorithm solving the Travelling Salesman Problem to the Qiskit Aqua Library. A Jupyter Notebook containing step-by-step instructions on the mathematics of the code will be provided along with examples of the acompanying code.

Methodology

The first step of the project will be to work through the algorithm mathematically on a small test cases to demonstrate the efficacy of the algorithm. Simple programs for computing the outputs of the matrix transformations may need to be written as the scale of some of the matrices may be too large to reasonably compute manually. The mathematics will be tested on small test cases between n=4 and n=6.

A classical brute-force algorithm to solve the TSP will be written to verify the results of the mathematical solutions as well as any experimental results received from the quantum algorithm. The brute-force method will execute in reasonable time at n < 15 – within the scope of any possible experimental test cases.

Next, one must setup an account within the IBM Q Experience. The IBM Q Experience provides the tools within its interface to create a Qiskit Jupyter Notebook. The Qiskit Aqua Library and API are written in Python (3.5 or later); IBM recommends using the Annaconda 3 distribution which includes all of the dependencies needed for Python to interact with Qiskit, including support for Jupyter Notebooks and NumPy.

This project will follow the following projected timeline:

* Month 1:
  + Mathematical analysis starts
  + Begin learning the necessary Python libraries for IBM Q
  + Implement classical brute-force TSP
* Month 2:
  + Conclude and confirm results from mathematical analysis
  + Begin implementing the quantum code for TSP
* Month 3:
  + Begin testing theoretical results of quantum algorithm
* Month 4:
  + Attempt to refine the quantum algorithm if results are subobptimal or optimizations in the time or space complexity are able to be made at this time.

Each step of the process will be documented in the Jupyter Notebook throughout the progression of this project, with the final results of the Notebook being available during Month 4. The quantum algorithm will be submitted to the Qiskit Aqua library when the code is found to be free of logic errors (any occuring errors are due to the physical limitations of the quantum computer, and not due to flaws in the logic of the code), and when the researcher has concluded that no further optimization of the code can be made within the allotted time for this project.

Anticipated Results

The primary results of this project are: 1. implementable quantum algorithm for solving the Travelling Salesman Problem that is free of logic errors, and 2. well-documented Jupyter Notebook that outlines the mathematics and logic of the algorithm. As the algorithm is meant to be applied by researchers of other disciplines, we expect the Notebook and the Algorithm to be accessible to users with limited background knowledge of computer science, mathematics, or quantum mechanics.

While optimizations on the current iteration of the Algorithm will be searched for and attempted, we do not, at this time, anticipate any optimizations to be made to the final implementation of the Algorithm.

References

[1] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Santa Fe, NM, USA, 1994, pp. 124–134.

[2] “The Golden Ticket,” *Princeton University Press*. [Online]. Available: https://press.princeton.edu/titles/9937.html. [Accessed: 15-Jul-2019].

[3] M. Held and R. M. Karp, “A Dynamic Programming Approach to Sequencing Problems,” *J. Soc. Ind. Appl. Math.*, vol. 10, no. 1, pp. 196–210, 1962.

[4] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Travelling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[5] M. Pistoia and J. Gambetta, “Qiskit Aqua — A Library of Quantum Algorithms and Applications,” *Medium*, 12-Dec-2018. .

[6] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi, “Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience,” *ArXiv180510928 Quant-Ph*, May 2018.

[7] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. New York [u.a: Freeman, 2009.

[8] A. M. Jaffe, “The Millennium Grand Challenge in Mathematics,” vol. 53, no. 6, p. 9, 2006.

[9] L. A. Hemaspaandra, “SIGACT News Complexity Theory Column 36,” p. 15.

[10] R. P. Feynman, “Simulating physics with computers,” p. 22.

[11] D. Deutsch, “Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer,” *Proc. R. Soc. Lond. Ser. Math. Phys. Sci.*, vol. 400, no. 1818, pp. 97–117, 1985.

[12] E. Bernstein and U. Vazirani, “Quantum Complexity Theory,” *SIAM J. Comput. Phila.*, vol. 26, no. 5, p. 63, Oct. 1997.

[13] C. H. Bennett, “Logical Reversibility of Computation,” *IBM J. Res. Dev.*, vol. 17, no. 6, pp. 525–532, Nov. 1973.

[14] E. National Academies of Sciences, *Quantum Computing: Progress and Prospects*. 2018.

[15] T. Albash and D. A. Lidar, “Adiabatic Quantum Computing,” *Rev. Mod. Phys.*, vol. 90, no. 1, p. 015002, Jan. 2018.

[16] J. M. Gottlieb, “Introduction To The Physics Of D-Wave and Comparison To Gate Model,” 2018.

[17] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *ArXiv180100862 Cond-Mat Physicsquant-Ph*, Jan. 2018.

[18] M. Amico, Z. H. Saleem, and M. Kumph, “An Experimental Study of Shor’s Factoring Algorithm on IBM Q,” *ArXiv190300768 Quant-Ph*, Mar. 2019.

[19] “Quantum devices & simulators,” *IBM Q*, 05-Jun-2018. [Online]. Available: https://www.research.ibm.com/ibm-q/technology/devices/. [Accessed: 15-Jul-2019].

[20] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC ’96*, Philadelphia, Pennsylvania, United States, 1996, pp. 212–219.

[21] Qiskit, *QISKit tutorials. Table of contents*. .

[22] Qiskit, “Qiskit Aer | A high performance simulator framework for quantum circuits,” 2019. [Online]. Available: https://qiskit.org/aer. [Accessed: 16-Jul-2019].

[23] Qiskit, “Qiskit Ignis | Understanding and mitigating noise in quantum devices.,” 2019. [Online]. Available: https://qiskit.org/ignis. [Accessed: 16-Jul-2019].

[24] Qiskit, “Qiskit Aqua | Algorithms for near-term quantum applications,” 2019. [Online]. Available: https://qiskit.org/aqua. [Accessed: 16-Jul-2019].

[25] “Project Jupyter.” [Online]. Available: https://www.jupyter.org. [Accessed: 16-Jul-2019].

[26] “The Jupyter Notebook — Jupyter Notebook 5.7.8 documentation.” [Online]. Available: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html. [Accessed: 16-Jul-2019].

[27] A. Ramanan, “Quantum computing at Microsoft.” [Online]. Available: https://blogs.msdn.microsoft.com/uk\_faculty\_connection/2018/02/06/introduction-to-quantum-computing/. [Accessed: 25-Jun-2019].

[28] J. Preskill, “Caltech Physics 219/Computer Science 219 Quantum Computation,” 2013. [Online]. Available: http://www.theory.caltech.edu/%7Epreskill/ph219/index.html#lecture. [Accessed: 22-Jun-2019].

[29] E. G. Rieffel and W. H. Polak, *Quantum Computing: A Gentle Introduction*. MIT Press, 2011.

Appendix A

Notation and Vector Representation of the Qubit

The state of a quantum computer is often represented with Dirac notation. The following ket-vectors: represent the 0 and 1 states (analogous to bits of 0 and 1 in a classical computer) of a single qubit within a quantum computing system. These states can be re-represented as vectors of the forms and , respectively.

To expand the system for multiple qubits, one takes the direct product of all qubits within the system. For example, a two-qubit system, with both qubits in state 0 is expressed by the following:



and in vector format:

. (3)

The associated bra-vector, or co-vector, for each state () is given as the transpose, row-vector, of the state with the complex-conjugate taken for each element. Therefore, we show the following:

.

Dirac notation typically omits the operator symbols when operating on two states within an equation. For clarification, we look at the three combinations of two states:

* implies direct product, as shown above;
* implies scalar product, such that:

; (4)

* implies cross product, such that:

. (5)

Qubit State

Because a qubit does not exist discretely in the states of or , we must find the probability that the qubit will be measured in either state. The probabilities are given by complex coefficients operating on either state, typically denoted α and β as scalars for the states and , respectively. Thus, we can describe the overall state of a single qubit given:

. (6)

We find the probability amplitudes that a qubit is measured as a 0 or 1 given the probability amplitudes of these coefficients:

[27] 

Appendix B

Quantum Gates

## Hadamard Gate

The Hadamard gate acts to superpose the state of a qubit, giving equal probability that the qubit is measured as either 0 or 1 [2].

* Hadamard Gate:

When applied to a qubit of either the or basis vectors:

; (8)

. (9)

Likewise, we can refer to the superposed state produced as the Hadamard basis vectors and :

(10)

(11)

Pauli-X Gate

The Pauli-X gate is the equivalent of a not-gate in classical computers, and acts to invert the values of the and states on a single qubit.

* Pauli-X matrix:

It is the case then that:

. (12)

Controlled Gates

Controlled gates require two qubits wherein one of the qubits acts as the control, while the other is being acted upon. A Controlled gate is notated as , where is the gate to be controlled [28]. In its generalized form, a controlled gate is constructed as:

[29] 

 

The effect is that the controlled gate only acts on the 2nd qubit if the control qubit’s state is 1, and has no effect if the control qubit’s state is 0.

The Controlled-Not gate (CNOT or ) then takes the form:

. (15)

This gate then maps the following two-qubit system as:

With the following as an example:

. (16)

Toffolli Gate

The Toffolli Gate (a Controlled-Controlled-Not) Gate has the following form:

.

Similar to the Controlled Not gate, the Toffoli acts to switch the 3rd bit iff the first and second bits are 1.