Grover's Algorithm and an Application in Hamilton Cycle Searching

Gupta, Ankur Smith, Avery Walsh-Costello, Maye Xu, Yichi

December 2020

Abstract

This paper offers an introduction of a quantum algorithm, namely Grover's algorithm, and it's potential use in NP-complete problem-solving. We demonstrate Grover's ability on a simplified NP-Complete Problem—finding a Hamilton cycle within a graph. This paper aims to provide a different manipulation of the problem and a clear presentation of the process of Grover's algorithm.

1 Introduction

Quantum computing has gained a lot of attention in recent years. As quantum hardware becomes faster and increasingly more accessible, many classical computing methods are suddenly becoming obsolete. Already, we know that quantum computers enable solving the integer factorization problem in polynomial time [6]. Many banks and financial institutions rely on the difficulty of the integer factorization problem in their cybersecurity systems. Quantum algorithms already pose a threat to cybersecurity as we know it and there is only more research being done into these algorithms as time goes on [3].

We explore a specific quantum algorithm—Grover's algorithm—and it's application on an NP-complete problem that isn't as malicious as breaking bank code: The Travelling Salesmen Problem (TSP).

2 Background

Before we can solve TSP with Grover's algorithm, we offer an introduction to quantum computing, some standard notation, and intuition on how use quantum mechanics to solve problems.

2.1 A Working Introduction to Quantum Mechanics

Richard Feynman once said,

> I think I can safely say that nobody understands quantum mechanics.

Quantum mechanics is founded upon a very simple theory, whose fundamental principles are concisely stated in four basic postulates. The full-detailed mathematical workings are out of the scope for this project; however, we recommend the reader to look into [2] for a

more formal explanation. Instead, we will give a quick overview of the quantum components relevant to Grover's algorithm.

2.1.1 Qubits

Quantum bits or *qubits* are the basic building blocks that encompass all fundamental quantum phenomena. They provide a mathematically simple framework in which to introduce the basic concepts of quantum physics into a computational framework. Recall that in the context of atomic physics, Heisenberg's Uncertainty Principle states that we cannot concurrently know both the position and the momentum of an electron [7]. Therefore, the electron is thought to be in every position and momentum combination at the same time.

Applying this concept to computing, we can represent this concept with a linear space. As we would like this space to be representative of a singular bit in a classical computer, let us limit it to two dimensions—0 and 1. Then, define the column vectors $|0\rangle$ and $|1\rangle$ to be the orthonormal basis for this state space.

The state of a qubit can be written as a unit column vector $\left[\begin{smallmatrix}\alpha\\ \beta\end{smallmatrix}\right]$ in this state space. In Dirac notation, this may be written as

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{with} \quad \alpha, \beta \in \mathbb{C} \quad \text{and} \quad |\alpha|^2 + |\beta|^2 = 1 \tag{1}$$

The real power of quantum computing appears when we consider a qubit as a space rather than a state. Rather, we consider the *superposition* of the qubit, in which the qubit is *every state at the same time*. A scalar value can then be extracted from the qubit via collapsing it, explained in section 2.1.2.

2.1.2 Measurement

Recall from Equation 1 that the qubit $|\phi\rangle$ is a linear combination of the basis $|0\rangle, |1\rangle$. As a qubit is a unit vector, a measure is simply *collapsing* the qubit into a single classical bit of information—0 or 1. This process fundamentally alters the state of the quantum system, as the result of the measurement becomes the new state.

To take a measurement, we first have to represent our current state as a linear combination of vectors which belong to orthogonal sub-spaces of our qubit. Let $|v\rangle$ be a qubit in the state v. Then, $|v\rangle$ can be represented as

$$|v\rangle = \alpha_0|v_0\rangle + \alpha_1|v_1\rangle + ... + \alpha_k|v_k\rangle \quad \text{with} \quad \alpha_0, \alpha_1, ..., \alpha_k \in \mathbb{C} \tag{2}$$

Because of our normalization condition, the sum of the magnitude of all α_i for $i = 0, 1, ..., k$ will equal 1. Then we can think about our qubit $|v\rangle$ taking state $|v_i\rangle$ with probability $\alpha_i\bar{\alpha_i}$. Now to take a measurement, we leave it up to chance to pick any state based on the probabilities, and return that state. This returned state now becomes the state we are working with and may be observed. The loss of information in taking a measurement means they need to be taken carefully or sparingly.

2.1.3 Gate

In the quantum circuit model of computation, a quantum logic gate is a basic quantum circuit operating on a small number of qubits. Quantum logic gates are represented by unitary matrices. The number of qubits in the input and output of the gate must be equal; a gate which acts on n qubits is represented by a $2^n \times 2^n$ unitary matrix. The quantum

states that the gates act upon are vectors in 2^n complex dimensions. The base vectors are the possible outcomes if measured, and a quantum state is a linear combination of these outcomes. As a side effect of the requirement that gates are unitary, all gate applications are reversible. Measurements however, are not (2.1.2).

2.2 Grover's Algorithm

We offer a high-level explanation of how Grover's algorithm works. Grover's algorithm is designed to search for an element in an unordered list of size N. In a classical computer model, it takes $O(N)$ iterations to complete as all N unordered elements need to be check. However, Grover's algorithm takes advantage of quantum mechanics to only take $O(\sqrt{N})$ to complete the same task.

2.2.1 Notation

In this section, we will introduce some essential notation and operators for understanding Grover's. As mentioned before, Grover's finds a singular element x out of a list of N elements.

The Oracle
In other words, we can define an *oracle* function upon this data set s.t.

$$f(e) = \begin{cases} 1, & e = x \\ 0, & e \neq x \end{cases}, \forall e \in N \tag{3}$$

With that oracle, let us define a unitary operator U_ω that behaves as the following:

$$\begin{cases} U_\omega|x\rangle = -|x\rangle, & f(x) = 1 \\ U_\omega|x\rangle = |x\rangle, & f(x) = 0 \end{cases} \tag{4}$$

In other words, if the oracle returns *true*, then the unitary operator U_ω will negate the state vector. Otherwise, the vector remains untouched.

Grover Diffusion Operator
While the exact theory is omitted due to the scope of this project, we denote $|s\rangle$ as the uniform superposition over all states as

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \tag{5}$$

When a gate operation is applied to a superposition, it changes the probability distribution of the qubit. The uniform superposition $|s\rangle$ above has equal probability applied to all possible states. Similarly, the *Grover Diffusion Operator* takes a probability distribution and normalizes it. While it is defined as

$$U_s = 2|s\rangle\langle s| - I \tag{6}$$

where I is the identity matrix, it is easier to think of as a normalization function–ensuring that all of the probabilities sum up to one. If you are familiar with machine learning, this is very similar to what softmax function does in classification problems.

2.2.2 Intuition & Algorithm Overview

Given these different parts, Grover's algorithm begins to form. The quantum nature of the algorithm means that whenever we apply a gate or a state change, we do so across the entire domain of our input.

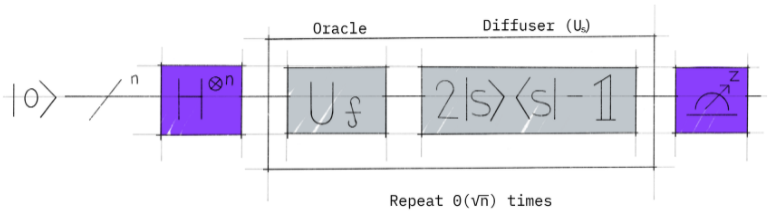


Figure 1: A gate overview of Grover's algorithm. The qubits are first put into superposition via the Hadamard gate. Then they are run through the oracle and the diffuser $\sqrt{N}$ times. Lastly, a measurement is taken to view the result.

Geometrically, we can view the oracle as reflecting our desired outcomes across the x axis as seen in 3. The diffusion gate then normalizes the superposition of qubits, boosting the probability of all reflected qubits in the process. Repeated applications of this reflection-boosting process will get us closer to the result that we desire following a measurement. This process is outlined in Algorithm 1. Observe that the system needs to be in superposition before any processing can be done. Putting a qubit into superposition is done via the well-defined Hadamard Gate $H\otimes$. The exact details are omitted due to the scope of this paper, but you can see how the Hadamard Gate plays into Grover's algorithm in Figure 1 the result of which shown in Figure 2.

Algorithm 1: Grover's Algorithm

Input: Oracle $f(x)$ and associated unitary matrix U_ω
Result: The unique integer $x^* \in [0, 2^n - 1]$ such that $f(x) = 1$
Program Initialization
Put the system into uniform superposition $|s\rangle$.
for $O(\sqrt{N})$ *iterations* **do**
  Apply the oracle operator U_ω
  Apply the Grover Diffusion Operator U_s
end

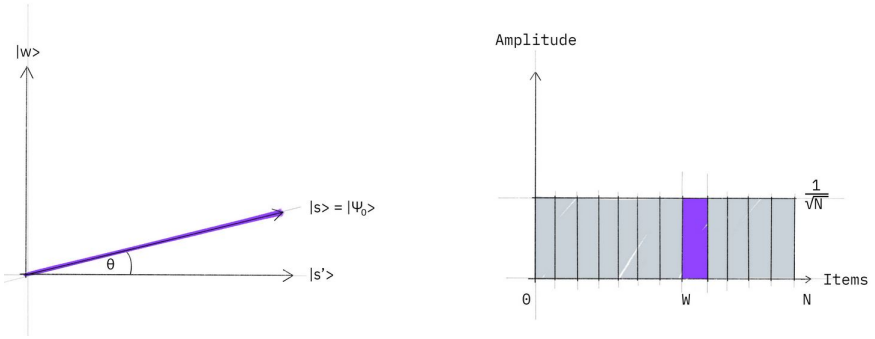


Figure 2: The qubits are placed in superposition with equal probability.

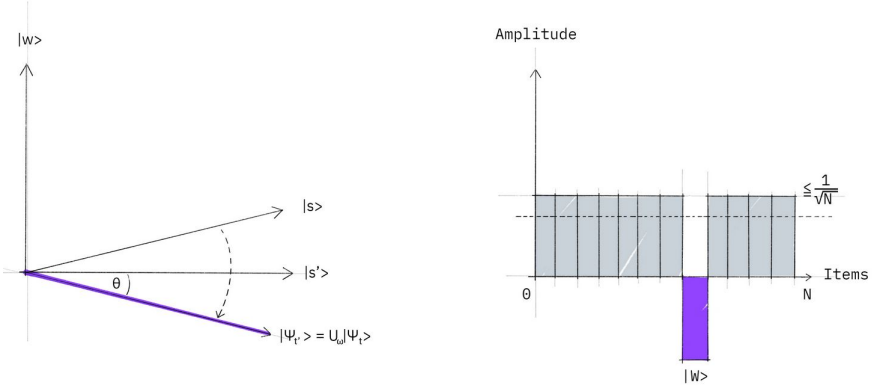


Figure 3: The item we are searching for, and only the item we are searching for is reflected. If no item is present, then no reflection occurs. Similarly, if many satisfactory items are present, they all are reflected.

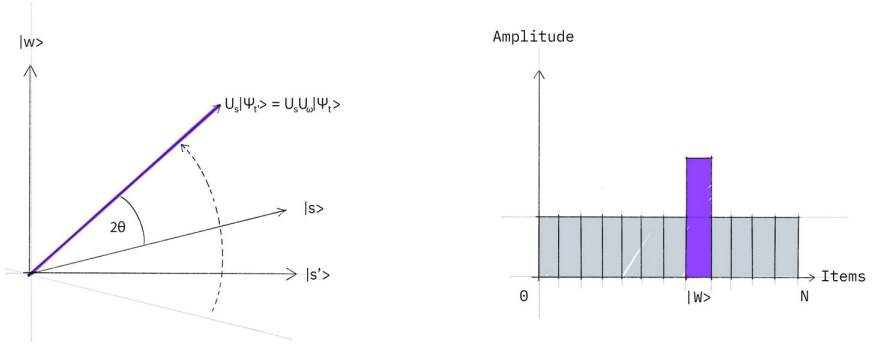


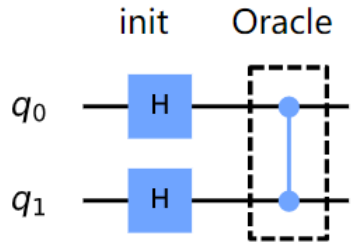
Figure 4: The values we are searching for are now amplified after the application of our diffusion gate. The probability we select the correct item during a measurement has been increased.

2.2.3 An Example on Two Qubits

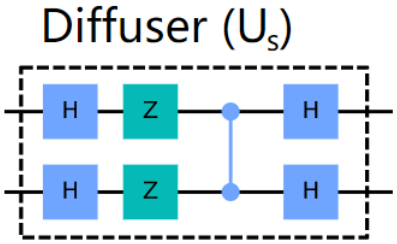
Let our domain be the possible values of two bits $\{00, 01, 10, 11\}$. Now we will walk though an example where we are trying to find the value 11. We represent 11 with the state $|11\rangle$ and we call this our *winner*, $|w\rangle$. The oracle U_ω for $|w\rangle = |11\rangle$ acts as follows:

$$U_w|s\rangle = U_w \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) \tag{7}$$

Observe how the polarity of $|11\rangle$ has flipped. The next step is encoding the oracle into quantum states. In this case, this is done via a controlled z-gate, as seen below.



Now that the oracle has been applied, we need to normalize the probabilities through the diffusion gate. The quantum circuit can be seen below:



Note that the Z-gate is a simple phase shift gate, defined as

$$Z(x) = \begin{cases} -1, & \text{if } 1 \\ 0, & \text{if } 0 \end{cases}$$

Putting everything together, we can make the final circuit, as shown in Figure 5. Note that in this case, since we have 2 qubits, that means that we can come up with a probabilistic solution in

$$\sqrt{2^2} = \sqrt{4} = 2$$

iterations.

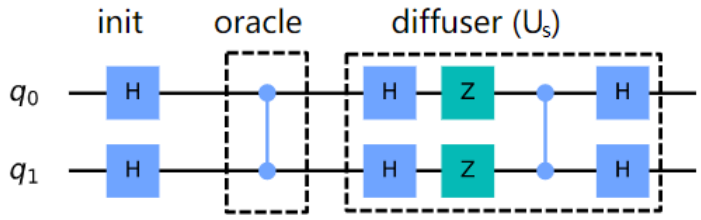


Figure 5: The quantum circuit for Grover's algorithm.

2.3 The Traveling Salesman Problem & NP-Completeness

As quantum computing allows the rapid sampling of a search space, it naturally seems to be applicable to NP-Complete problems. *NP-Complete* is a class of problems where finding a solution takes exponential time, but verifying a solution is quite easy. This idea of an easy to verify solution leads directly to usage in an oracle. One of these problems is the Travelling Salesman Problem (TSP), in which we try to find a Hamilton cycle within a graph [4].

In graph theory, a Hamilton path (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once. A Hamilton cycle (or Hamilton circuit) is a Hamilton path that is a cycle. An example of such a cycle can be see in Figure 6.

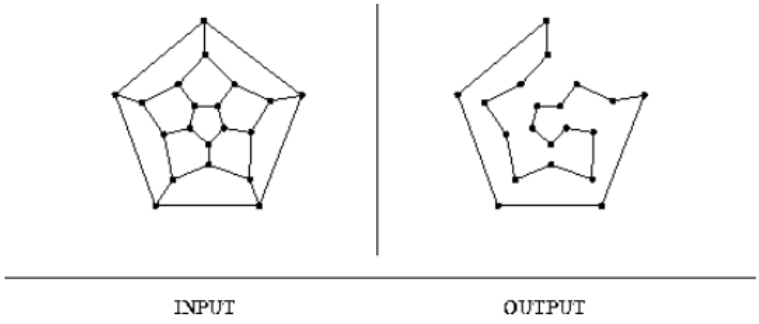


Figure 6: An example of a Hamiltonian Cycle within a graph.

3 Our Work

Now that we have introduced the relevant background for our project, we can begin to discuss our contributions.

3.1 In Search of a Hamilton Cycle

We set out to convert a typical discrete optimization problem into one that could be run through Grover's algorithm. We decided to use the question: "Does the given graph G contain a Hamilton cycle?"

In order to have this question be compatible with Grover's search, we need three things:

1. An unordered list to search through. We call this list the *domain* and we will denote it as D. The size of the domain we denote N ($N := |D|$).

2. An element $a \in D$ that we are searching the domain for.

3. A function $f(x)$ where $f(a) = True$ (a is the element we are searching for from above), and $f(b) = False \; \forall b \in D$, where $b \neq a$. This function will be called the *oracle*.

So our question now becomes: *"Given a graph $G = (V, E)$ and a Hamilton cycle C, is C in G?"*

Notice that there are no constraints on what C could belong to other than that it is a Hamilton cycle. C could belong to a different graph, but we want to check it against our graph G all the same. This poses a problem though, if we cannot find a Hamilton cycle efficiently, we cannot give it as an input to Grover's algorithm. Grover's search has a way around this. As long as we have a way to verify solutions across our entire domain, we can use this in Grover's algorithm to find our solutions anyways. Our function then becomes $f(a) = True$ where $a \subseteq E$ and a is a Hamilton cycle and $f(b) = False$ where $b \subseteq E$ but it is not a Hamilton cycle. This function operates in polynomial time, and it covers our domain so it will work nicely. Now the question we pose: *"Given a graph $G = (V, E)$ and a function $f(x)$ that verifies if a given $x \subseteq E$ is a Hamilton cycle for G, can we find a Hamilton cycle?"*

3.2 Qiskit Simulations

IBM provides an open source quantum computing framework called Qiskit [1] to run quantum circuits. Qiskit offers packages for Python that can be used to either simulate and draw a quantum circuit on a classical machine, or request a task be completed on a real quantum computer. We utilized the packages to create a demo of our algorithm. The full code can be found at

`https://github.com/agupta231/ma3233-final-project`

3.3 Creating the Oracle

Our first task was to build our function that would become our oracle.

We created a classical function that takes a cycle denoted by edge pairs and returns whether it is a Hamilton cycle. This function can be seen in Figure 7.

Next, we need to turn this function into an oracle that can be utilized by Grover's Algorithm. There are two methods to this task: recreate the function using a combination of quantum gates, or create a truth mapping for our domain.

Since quantum gates are represented by unitary matrices, we can convert our function into a matrix transformation that gives the desired results. Quantum kickback involves taking the matrix representation of the function $f(x)$ and sending it to an output of $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ which will give the desired oracle gate for our usage.

This effect of quantum kickback is out of scope of our project, but a brief explanation of the results of it is given and a gate visualization can be seen in Figure 8. This method happens in one quantum operation, and is the desired way to handle conversions of functions to Oracles. Unfortunately, it requires access to a quantum computer in the first place to turn our linear transformation into a quantum gate. Additionally, we need to convert our

```python
def is_hamiltonian(self, edges: List[t.edge]) -> bool:
    """Check if the edges make a hamiltonian cycle.

    Args:
      edges (List[t.edge]): the edges which make up the proposed cycle

    Returns:
      bool: Whether or not the cycle is hamiltonian
    """
    # Edge Count constraint
    if len(edges) != len(self.vertices):
        return False

    # Vertex Saturation constraint
    included_vertices = {v for e in edges for v in e}
    if included_vertices != self.vertices:
        return False

    # Connectedness & cycle constraint
    # Create a dictionary from vertex -> (active_degree, "leader")
    vertex_meta = {v: [0, v] for v in included_vertices}
    for e_from, e_to in edges:
        leader = min(vertex_meta[e_from][1], vertex_meta[e_to][1])

        vertex_meta[e_from][1] = leader
        vertex_meta[e_from][0] += 1
        vertex_meta[e_to][1] = leader
        vertex_meta[e_to][0] += 1

    first_leader = next(iter(vertex_meta.values()))[1]
    for degree, leader in vertex_meta.values():
        if degree != 2 or leader != first_leader:
            return False
    return True
```

Figure 7: A classical Python3.8 function to determine whether the given edge list is a Hamilton cycle for the graph

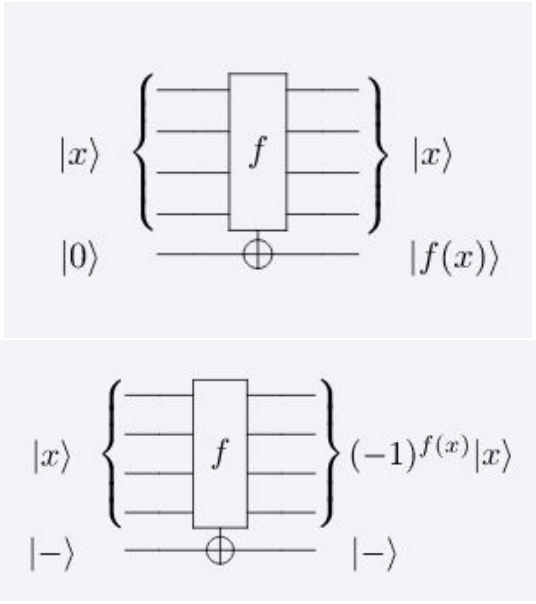


Figure 8: The gate representation of quantum kickback

function into a linear transformation $A\mathbf{x} = \mathbf{b}$. This process is not trivial and would take an extensive amount of work, so we decided to go the route of truth mapping our domain.

As stated above, the oracle is a matrix transformation. Specifically, it takes the form:

$$U_f = \begin{bmatrix} (-1)^{f(x_0)} & 0 & ... & 0 \\ 0 & (-1)^{f(x_1)} & ... & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & ... & (-1)^{f(x_n)} \end{bmatrix} \tag{8}$$

This computes the oracle against every possible value of $x \in D$ and raises -1 to the power of the result of the function against x. A $True$ value corresponds to 1 and a $False$ value corresponds to 0. All true entries will be -1 and all false entries will be a positive 1. The result of a transformation of the oracle on any qubit state in the domain will give:

$$U_f|x\rangle = \begin{cases} |x\rangle & \text{if } f(x) = False \\ -|x\rangle & \text{if } f(x) = True \end{cases} \tag{9}$$

This obtains the reflection effect we desire. The quantum kickback method creates this oracle in one step, but we can also recreate this matrix manually using a classical method. Because recreating it classically is relatively simple, it was our choice of method. An important note: this method does have significant drawbacks outside the research setting. Any speedup that would be gained from Grover's algorithm is lost when manually creating the oracle map. To do so, you need to find all the answers to every input in the domain. The time required to do such a thing takes 2^n operations! At this point, we have essentially already answered the question. After all, any input with a corresponding -1 entry satisfies our problem. But, we can still examine the complexity and time usage of Grover's algorithm independently, as Grover's does all of this in one operation.

3.4 Running the Algorithm

The `generate_truth_table` code block is actually doing two things. It is creating our domain alongside with the oracle. In order to save time for Grover's algorithm later, we limited the domain as much as we could. We know that Hamilton cycles must have an equal number of edges as there are vertices, so we only generated $\binom{|E|}{|V|}$ edge combinations. This is now the size of the domain of our function, and constitutes the size of the "list" that we are searching through.

Grover's algorithm will then run using our oracle over our domain and output a result. Each qubit of the result represents a specific combination of edges in the graph. The superposition of qubits in the registers gives the probability that any one qubit will be selected after measurement, so the qubits with the highest probabilities correspond to the edge combinations that are Hamilton cycles, unless there is no Hamilton cycle anywhere in the graph. We take the measured qubit and check it against our oracle one last time. If the combination works against our oracle, we know we found a solution. If it does not, we can either run Grover's algorithm again and hope for a better result, or state that we could not find a solution. In our program, we simply take the latter choice.

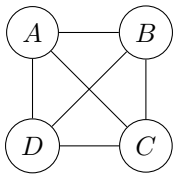
3.5 Test Trials

Once we had Grover's implemented, we ran it on a suite of test cases to see how it would interact with a real-world problem. Recall that to optimize our runtime, we calculated the domain of $\binom{|E|}{|V|}$ possible cycles *a priori*. In the test trials below, the measurement charts have binary numbers on the bottom with their associated probabilities.

These binary numbers are the index of a particular edge combination in base 2. This might be easier to understand with an example. Assume that we have domain D size N. That means that D's indices are $[0, N-1] \cap \mathbb{Z}$, where each entry corresponding to an edge combination within the graph. Thus, value `010` would correspond to the potential cycle at $D[010_2] \implies D[2]$.

3.5.1 K_4

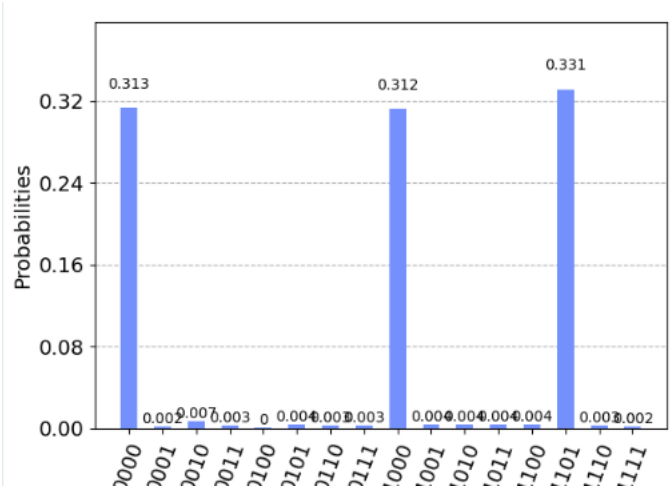
Our base test case was K_4, due to how well we understood it:



Observe that in this example, there are three Hamilton cycles:

1. $\{(A,B),(B,C),(C,D),(D,A)\}$
2. $\{(A,B),(B,D),(C,D),(C,A)\}$
3. $\{(A,D),(B,D),(B,C),(C,A)\}$

After we ran Grover's and collected our measurement, we get the following probability distribution:

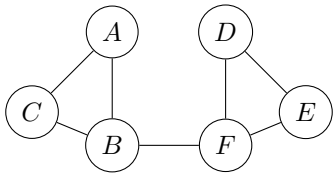


As observed in the measurement above, we can see three clear peaks in the distribution, all with similar probabilities and significantly higher than the rest of the possible solutions. Once we convert these binary representations to base 10 and index the domain, we find out that these three peaks exactly correspond to the three Hamiltonian cycles.

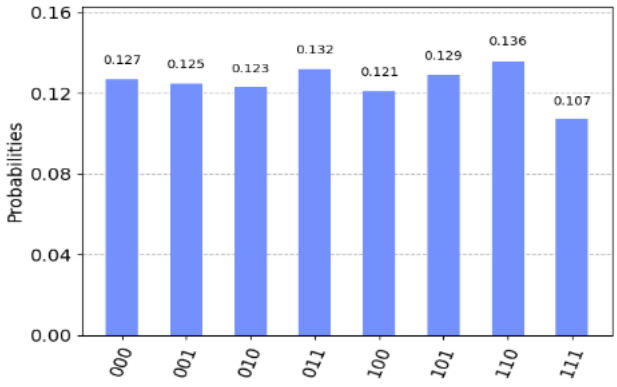
Therefore, Grover's didn't only find a Hamiltonian cycle within the graph, but rather found *all* of the Hamiltonian cycles in the graph.

3.5.2 6 Vertices; No Hamiltonian Cycles

We also ran Grover's on the negative test case of a graph with no Hamiltonian cycles:



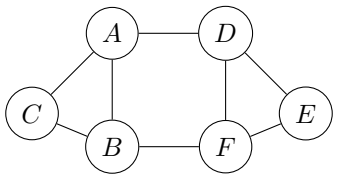
Clearly, there are no Hamiltonian cycles in this graph. Then looking at the quantum measurement:



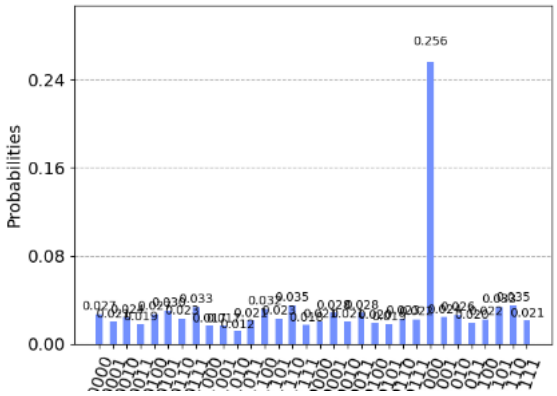
Here, we can't find any discernible peaks, so we are led to conclude that there are no Hamiltonian cycles, which is the correct solution.

3.5.3 Single Hamiltonian Cycle

For our final example, we took the graph in Section 3.5.2 and added a singular edge to it in order to induce a singular Hamiltonian cycle:



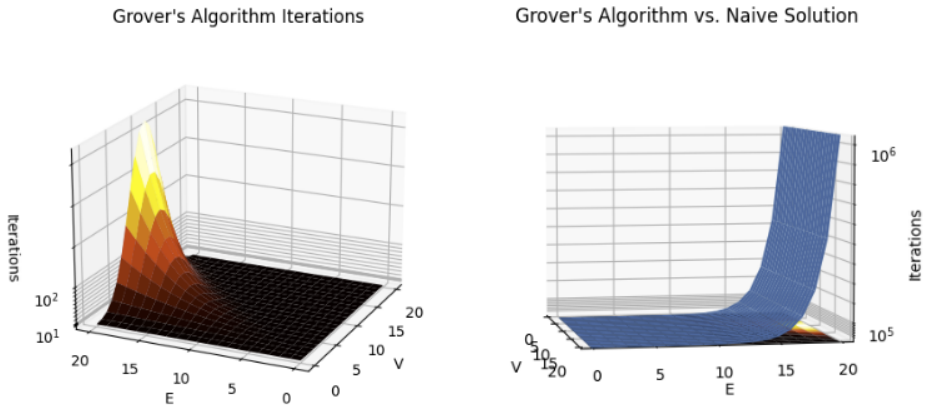
Here the Hamiltonian cycle is clearly the outer perimeter of the graph. Looking at the quantum measurement:



We can see a singular peak for one of the choices. When we find the binary representation's associated cycle, we find that Grover's correctly identified the Hamiltonian cycle.

4 Conclusion

We have demonstrated that Grover's algorithm is able to find Hamiltonian cycles within a graph. Grover's algorithm uutilizes quantum superposition to apply the search effect on each possible combination at the same time. So, we have been able to reduce the runtime of our solution. As mentioned in Section 2.2, we know that Grover's can search a list of size N in $O(\sqrt{N})$ time and that the size of our domain is $\binom{|E|}{|V|}$ from Section 2.3.



(a) The iterations graphed for Grover's algorithm as a function of $|V|$ and $|E|$

(b) Grover's algorithm's runtime vs. a classical runtime of $O(2^n)$

Figure 9: A runtime comparison between Grover's aglorithm and a classical algorithm. That is not a misprint; Grover's algorithm's runtime almost looks like plane when compared to a classical algorithm.

Therefore, we find that Grover's final runtime is $O\left(\sqrt{\binom{|E|}{|V|}}\right)$, which is a large improvement over the classical solution. The comparison between the two runtimes can be seen in Figure 9. Our results show that Grover's algorithm offers a significant improvement over a classical algorithm in finding a Hamiltonian cycle within a graph.

While it is true that we precompute the entire result set (effectively solving the TSP in a classical manner), we believe our results hold true. Consider the following proof. In the ideal case, if we encoded our Hamiltonian cycle test into quantum gates, we would be required to validate if a cycle was Hamiltonian or not within one execution of the quantum circuit. Therefore, we know that our quantum Hamiltonian test would have runtime $O(1)$. In our case, as we already precomputed all of the possible solutions, our "oracle" is really just performing a trivial hashmap lookup, which also has runtime $O(1)$. Thus, as the runtimes are the same between the two oracles, Grover's runtime remains unchanged, meaning that all of our results are valid.

However, we haven't solve $P = NP$ quite yet. The asymptotic upper bound remains exponential as the graph becomes dense. In our particular cases, we only gave Grover's algorithm undirected graphs with edge weight 1. Yet, TSP is often used to solve problems such as road routing, where the graphs are much more dense, directed, and most definitely have edge weights (traffic, distance, etc). So while our implementation of Grover's isn't applicable to the real-world *just* yet, it shows quantum computing's promise in revolutionizing computing as we know it.

5 Future Work

Searching for a Hamilton cycle is an NP-complete problem; the finding of a polynomial time algorithm, even with the power of quantum computing, would be a tremendous breakthrough. But, there does remain a seemingly large room for improvement on the upper

bound of the quantum search. Clever ways to reduce the search domain, or guaranteeing a certain amount of Hamilton cycles, would indicate a lowered upper bound. In the unweighted case, introducing the concept of isomorphism could significantly lower the domain by eliminating edge combinations that are essentially identical, but it would also reduce the possible Hamilton cycles to 1, as any Hamilton cycle is isomorphic to the other. While the reduction of possible solutions may seem like a minor point, we left out an interesting piece of information. If the domain of size N has M solutions, the runtime of Grover's algorithm is actually $O(\sqrt{\frac{N}{M}})$. This opens up many possibilities of examination. The number of non-isomoprhic Hamilton cycles in a complete graph is a known value. A deeper investigation into critical points in relation to the ratio of edges to vertices could provide promising results for a further reduction in iterations.

Travelling Salesman Problems often are applied to the case of weighted or even directed graphs. A first extension of our work would be to look at ways to solve the cases of finding a minimum weight Hamilton cycle in the graph. This poses an extra set of challenges, especially in the construction of the oracle. To determine if a cycle is minimum weight, we would need to know all of the other possible Hamilton cycle weights as well, or come up with a clever way to determine if a Hamilton cycle is minimum weight. Simply running our version of Grover's algorithm multiple times and checking against our different outcomes will not work, as we could have three solutions and repeatedly get unlucky and only have one returned back consistently.

Our method could also be applied to other graph searching problems by swapping out the oracle function. By changing the oracle to verify whether a graph is a tree, this method can then be applied to searching for a tree in a graph. Polynomial time algorithms already exist for this problem, so it would require more research just to get our method on par with Kruskal's or Breadth-First-Search.

Additionally, quantum computers have yet to find a full footing in everyday usage. Modern quantum computers are limited in the number of qubits they can use. As of this writing, the largest readily available quantum computer is IBM's `q_qasm_simulator` which only has 32 qubits. Unfortunately, this means that the time saved through a quantum algorithm cannot be obtained for large inputs, as no way to run the algorithms in pure quantum form exists. IBM hopes to make large breakthroughs in the quantum computing area, with plans for a 1000-qubit quantum computer in the near future [3].

6 Acknowledgements

Thank you to (hopefully soon to be Dr.) James McClung for meeting with us to give the groundwork for what would be in and out of scope of this project [5]. Thank you to Kyle Dituro for also helping with the initial step of getting the project off the ground. Thank you to Dr. Brigitte Servatius for leading the course in MA3233 Discrete Optimization. Lastly, thank you to Dr. William Martin for devising the notes that have become the text of Discrete Optimization [4].

References

[1] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis,

Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. Qiskit: An Open-source Framework for Quantum Computing, January 2019. Language: en.

[2] Lauren M. Baker and Liam M. Ogren. Quantum Algorithms from a Linear Algebra Perspective. Technical report, Worecester Polytechnic Institute, April 2017.

[3] Adrian Cho. IBM promises 1000-qubit quantum computer—a milestone—by 2023. *Science*, September 2020.

[4] William J. Martin. Discrete Optimization, September 2020.

[5] James McClung. personal communication, November 2020.

[6] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997. arXiv: quant-ph/9508027.

[7] Wikipedia contributors. Uncertainty principle — Wikipedia, the free encyclopedia, 2020. [Online; accessed 9-December-2020].