Quantum Walk Algorithm

PHY 612: Introduction to Quantum Information Science and Quantum Technologies

Iqra Imran and Fatima Aklaq

2021-12-0011 and 2020-12-0006

April 28, 2023

Abstract

The Quantum Walk Algorithm is in principle a search algorithm primarily used to search for marked vertices in a graph. Quantum Walks are motivated by the Classical Markov Chains (classical random walks) but there is nothing random in Quantum Walks. The Quantum walks algorithm provides a quadratic speedup in computations in comparison to its classical counterpart by employing the power of superposition. In this project, we will provide a brief introduction to classical Markov chains to draw an analogy for a quantum walk and then introduce the concept of coin space and coin operator which decides each step of the walker. After that, we would look into the mathematical formulation of the algorithm and implement it on a 4-dimensional hypercube. The circuitry of the algorithm varies from case to case, in this project we would implement it for searching marked indices on a hypercube.

Contents

1 Introduction

Let us first understand what a random walk means. Random walk as evident from the name; is a stochastic process. For example, a particle lives on an integer line and wants to move from one position to another on the number line. In a random walk, he would take each step randomly either going forward or backward. The probability that it would go forward or backward is $\frac{1}{2}$. It would continue moving in this fashion until it reaches the desired location. This is a probabilistic process.
Markov chains are a type of random walk on a graph. In Markov Chains the next state of the chain depends only on the current state—it is not influenced by the past states of the walker. The next state is determined by some deterministic or random rule based only on the current state[3]. In discrete time Markov Chains, the time steps are discrete. Let's consider the example of a graph G with V vertices and N edges. The vertices are often referred to as states and the edges are referred to as transitions between those states. With each edge, there is an associated probability of transition between the two states.

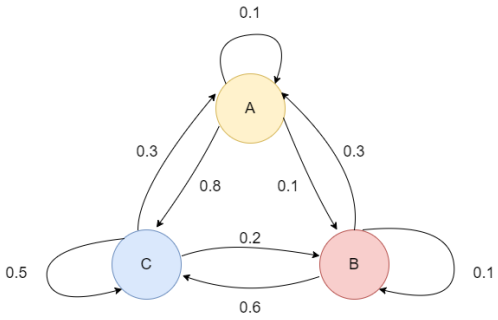


Figure 1: A complete graph with 4 nodes[4]

If we have a complete graph(in which all nodes are connected to every other node), we can represent the probability densities in terms of a transition matrix P, which has dimensions of $N \times N$ if the graph contains N vertices/nodes.

$$P = \begin{pmatrix} 0.1 & 0.3 & 0.3 \\ 0.1 & 0.1 & 0.2 \\ 0.8 & 0.6 & 0.5 \end{pmatrix} \tag{1.1}$$

This transition matrix stores all the information about where the particle is most likely to go after a step. If the walker takes t steps then simply the probability distribution can be achieved by applying P^t on the initial state of the graph.
Quantum walks are the quantum equivalent of the classical Markov chains. The key difference between both is that in the quantum version, the phenomena of superposition take place and instead of going from one state to the other state the walker assumes a simultaneous superposition over all states. This superposition continues to evolve until we make the measurement. In classical random walks measurement is performed at each and every step but in quantum walks Due to quantum interference, some states will cancel out. This makes quantum walk algorithms faster than classical ones since we can design them in such a way that wrong answers quickly cancel out.[4].
There are two types of quantum walks: discrete-time quantum walks and continuous-time quantum walks. In this project, as we are dealing with graphs so we would explore discrete-time quantum walks which are further divided into two categories: Coined quantum walks and Szegedy quantum walks. Coined quantum walks take place on the vertices of the graph whereas Szegedy quantum walk takes place on the edges of the graph.[4] In literature, it has been proven that both are equivalent in the case of a Grover coin. Here we would describe the coined quantum walk and then we would see how it can be implemented on a hypercube.

2 Mathematical background

2.1 Coined quantum walk

We have seen that in a classical random walk, the update mechanism for each step is a random process for example a coin toss. In the case of quantum walks the update decision must create a superposition position over states. For this purpose many coins have been introduced, for example, Hadamart and Grover coins. In the coin model, we have two quantum states and two operators. The first state is the position state, which represents the walker's position. The other state is the coin state. The coin state decides how the walker should move in the next step[4]. Both states can be represented by vectors in the Hilbert space $\mathcal{H}$. The position state is a part of the space $\mathcal{H}_P$ and the coin state is a part of the space $\mathcal{H}_C$. Thus the whole quantum state for the walker is the tensor product of both spaces i.e $\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_P$.
Similarly, this model has two operators; the coin operator, $\hat{C}$ and the shift operator, $\hat{S}$. $\hat{C}$ acts on $\mathcal{H}_C$ and put the walker in a superposition. The most common ones are the Hadamart operator and Grover's operator. The Hadamart operator puts the walker in an equal superposition whereas Grover's operator works a bit differently. It does create a superposition but the probabilities are not equal for all states. The probability amplitude of the current state of the walker is higher than the probability for any other state. In Figure 2, the current state of the walker is $|000\rangle$.

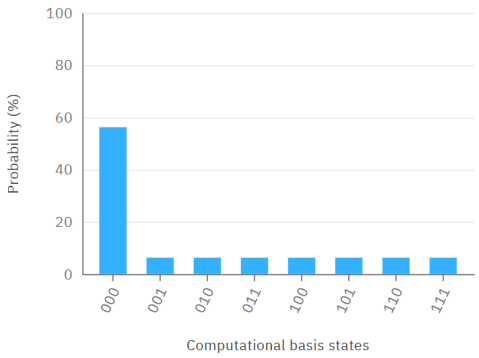


Figure 2: Probability distribution of a superposition created by Grover's coin[4]

The other operator is the shift operator $\hat{S}$ which acts on $\mathcal{H}_P$. It shifts the walker to the neighboring nodes upon the decision made by the coin operator. Thus shift operator is a controlled operation. One step of the random walk includes both the coin and shift operator. The next step of the walker is controlled by a complex amplitude rather than a probability. This generalization, from positive numbers (probabilities on classical Markov chains) to complex numbers, makes quantum walks more powerful than classical random walks[2]. Therefore we can define a unitary operator for the coined walk as $\hat{H}$ as follows:

$$\hat{U} = \hat{S}\hat{C} \tag{2.1}$$

If the walker takes t steps then the evolution of its initial state $|\psi(0)\rangle$ can be written as:

$$|\psi(t)\rangle = \hat{U}^t |\psi(0)\rangle \tag{2.2}$$

Where $|\psi(t)\rangle$ represents the state of the system after t steps[3].

3 Quantum Walk Algorithm

3.1 Mathematical Formulation

Quantum Walks are widely used to find marked vertices of a graph. This algorithm performs a quantum walk to find the marked vertices on a graph and later we will see its implementation of finding marked vertices on a hypercube. First, it must be noted that quantum mechanics restricts us to preserve the norm i.e. it demands unitarity. Thus the transition matrix P transforms into a norm-preserving matrix $W(P)$ in the quantum walk algorithm. The basis state of a classical random walk is the current vertex we are at, while in a quantum walk a basis state of has two registers, the first corresponding to the current vertex and the second corresponding to the previous vertex[1]. We can define a superposition over all vertices of the neighbors of the x-node as:

$$|p_x\rangle = \sum_y \sqrt{P_{xy}}\,|p_y\rangle \tag{3.1}$$

Our basis state would be $|x\rangle\,|y\rangle$, we would refer to it as a "good" state if **x** is marked otherwise it would be a "bad state". Say we have M marked vertices out total N vertices of the graph. Then we can define a "good" and "bad" state as a superposition over basis states.

$$|G\rangle = \frac{1}{\sqrt{|M|}} \sum_{x\in M} |x\rangle\,|p_x\rangle \quad , \quad |B\rangle = \frac{1}{\sqrt{N-|M|}} \sum_{x\notin M} |x\rangle\,|p_x\rangle \tag{3.2}$$

If $\epsilon = \frac{|M|}{N}$ and $\theta := \arcsin(\sqrt{\epsilon})$ then the uniform state over all the edges of the graph can be written as:

$$|U\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle\,|p_x\rangle = \sin\theta\,|G\rangle + \cos\theta\,|B\rangle \tag{3.3}$$

Now the algorithm for searching a marked vertex if an ϵ-fraction of nodes are marked is as follows [1]:

1. Setup the starting state $|U\rangle$
2. Repeat the following steps $O(\frac{1}{\sqrt{\epsilon}})$ times
   (a) Reflection through $|B\rangle$
   (b) Reflection through $|U\rangle$
3. Measurement to check whether the resulting vertex is marked or not.

Step 1 can be easily implemented by creating a superposition through Hadamart gates. Let us look at how the reflections take place.

3.1.1 Reflection through $|B\rangle$

Reflection through $|B\rangle$ is simple: we just have to recognize that if the first register contains a marked index **x** then change its sign. This can be achieved by using a phase oracle that shifts the phase if **x** is marked and unaltered otherwise [4].

3.1.2 Reflection through $|U\rangle$

This is where the quantum nature of the walk kicks in. The quantum analog of P is a unitary $W(P)$. To see how the reflection through $|U\rangle$ is implemented let us explore the evolution operator $W(P)$. As it is unitary its eigenvalues can be written as $e^{\pm 2\iota\theta_j}$. The state $|U\rangle$ is the eigenvector of $W(P)$ with eigenvalue-1 i.e. $\theta_j = 0$.

The reflection through $|U\rangle$ is equivalent of implementing a reflection $R(P)$ through the subspace spanned by the eigenvalue-1 eigenvectors of $W(P)$ i.e. $R(P)$ provides the following mapping[4]

$$|U\rangle \mapsto |U\rangle \tag{3.4}$$

and

$$|\psi\rangle \mapsto -|\psi\rangle \,\forall\, |\psi\rangle \text{ in the span of eigenvectors of } W(P) \text{ that are orthogonal to } |\psi\rangle \tag{3.5}$$

We will implement $R(P)$ by using phase estimation with the precision of $O(\frac{1}{\sqrt{\delta}})$ on $W(P)$ to distinguish the eigenvalue-1 eigenvectors from the other eigenvectors by adding a register with auxiliary qubits, where δ is the spectral gap of P. This precision requires $O(\frac{1}{\sqrt{\delta}})$ applications of W(P), and $O(log(\frac{1}{\delta}))$ auxiliary qubits that start in $|0\rangle$ and where the estimate will be stored. Phase estimation always gives an estimate $\tilde{\theta_j}$ of θ_j that is within a precision $\frac{\sqrt{\delta}}{2}$. We then multiply the state with a -1 if the estimate is sufficiently far from 0, and finally reverse the phase estimation to put the auxiliary qubits back to 0. Let $|w\rangle$ be an eigen vector of $W(P)$ with eigen value $e^{\pm 2\iota\theta_j}$ [1][4]. The map provided by $R(P)$ is:

$$R(P) : |w\rangle\,|0\rangle \xmapsto{PE} |w\rangle \left|\tilde{\theta_j}\right\rangle \mapsto (-1)^{[\theta_j \neq 0]}\,|w\rangle \left|\tilde{\theta_j}\right\rangle \xmapsto{PE^{-1}} (-1)^{[\theta_j \neq 0]}\,|w\rangle\,|0\rangle \tag{3.6}$$

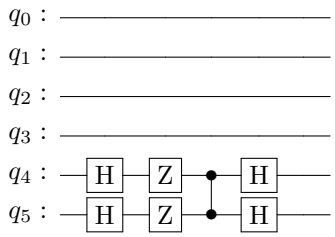
This has the desired effect: ignoring the auxiliary qubits (which start and end in 0), $R(P)$ maps eigenvectors with eigenvalue-1 of $W(P)$ to themselves, and puts a -1 in front of the other eigenvectors[1]. This is how we implement the quantum walk algorithm.

3.2 Implementation on 4D Hypercube

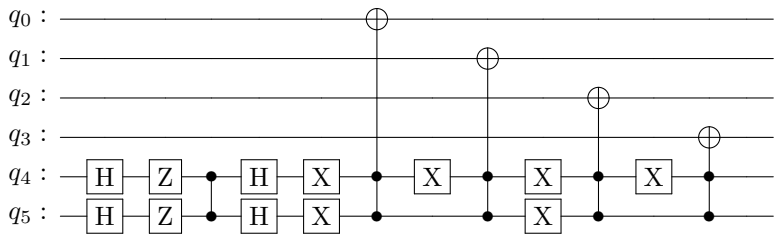
A hypercube is an N-dimensional generalization of a 3D cube. There is 2^N number of nodes in a hypercube. The nodes are represented by binary numbers to ease commuting. In this algorithm, we would search for a marked node/vertex on a 4D hypercube. Every node is connected to every other node whose binary representation only differs by one digit. For example, 1111 is connected to 1110, 1101, 1011, and 0111. As mentioned in the previous section, the Hilbert space for a quantum walker is $\mathcal{H} = \mathcal{H}^N \otimes \mathcal{H}^{2N}$, where $\mathcal{H}^N$ represents the coin space and $\mathcal{H}^{2N}$ represents the walker's position. The unitary operator which drives the algorithm constitutes of two operators:

$$\hat{U} = \hat{S}\hat{C} \tag{3.7}$$

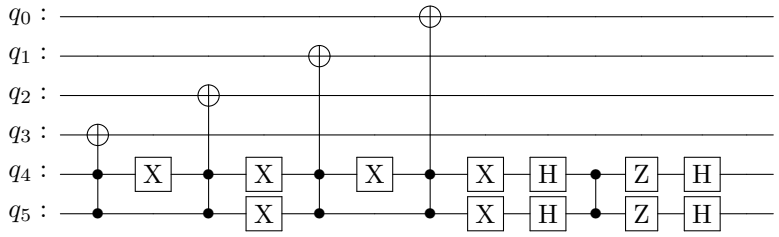
Let us first devise the circuitry for these two operators. We would use Grover's coin in this algorithm which uses 2 additional qubits along with the 4 qubits representing the walker's position and nodes on the hypercube.



After the coin operator, the walker will shift its position by implementing the shift operator. The shift operator is constructed by using NOT and CCNOT gates on one of the qubits of the walker's current position. If the coin's qubits have the state $|11\rangle$, we move the walker to the state in which the first node qubit differs. If the coin is $|10\rangle$ or$|01\rangle$, the walker moves to the state where the second and third qubits, respectively, differ. Finally, if the Grover coin is $|00\rangle$, we flip the fourth qubit[4].

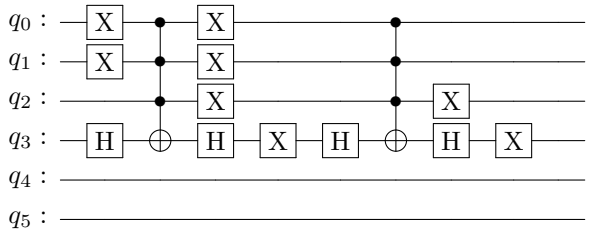


For implementing we require the inverse of the above operation $\hat{U}$ as well which is as follows:



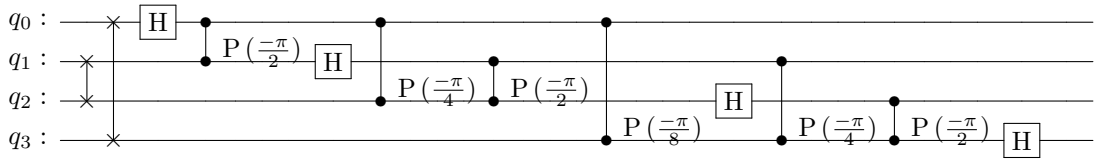
3.2.1 Phase Oracle

After creating superposition through Hadamart gates, We need to mark the indices we are searching for on the hypercube. Let us say we want to search for $|1100\rangle$ and $|0011\rangle$. We would use the following circuitry for step 2(a). As mentioned in the previous section in order to implement step 2(a) we require a phase oracle that marks the desired nodes.



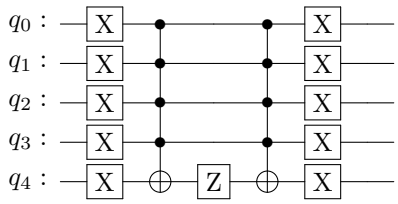
3.2.2 Phase estimation

After marking the desired nodes the actual fun part comes in which is using phase estimation to carry out a reflection through state $|U\rangle$ in step 2(b). As we know we require Inverse Quantum Fourier Transform circuitry and Quantum Fourier Transform to carry the phase estimation and its inverse respectively.



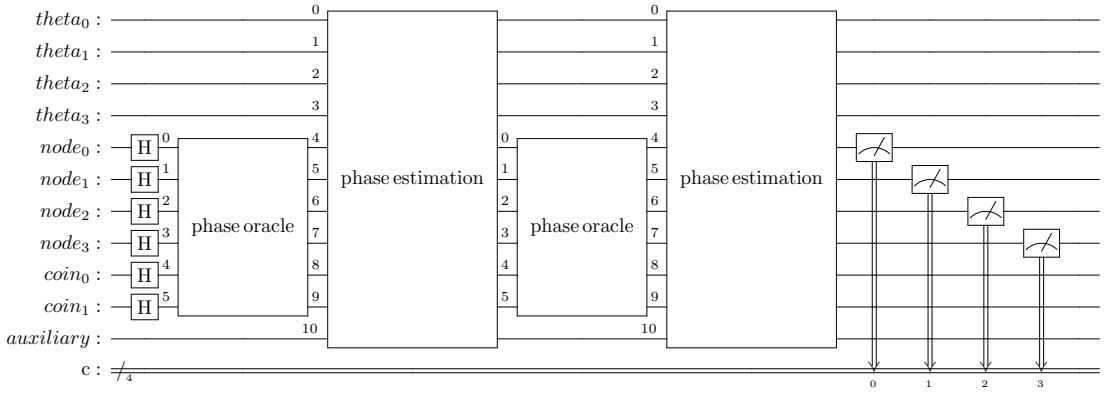
In phase estimation, we will rotate the auxiliary qubit if $\theta \neq 0$ which implies that we rotate an auxiliary qubit if the other qubits are non-zero. The gate required for marking this qubit with a negative sign is

referred to as "mark-auxi-gate" and is as follows:



Now, we have all the necessary gates to implement step 2(b). This step consists of a phase estimation one step of the quantum walk followed by an auxiliary qubit that we rotate if $\theta \neq 0$. For this purpose, we employ the mark-auxi-gate that we just created. Afterward, we reverse the phase estimation.

This completes all the parts of step 2 of the algorithm. Now let us summarize the whole algorithm as



3.3 Results

Finally, we run the implementation on the qasm simulator. We can see that the majority of the time the circuit collapses to the marked state $|0011\rangle$ and $|1100\rangle$ of the hypercube.

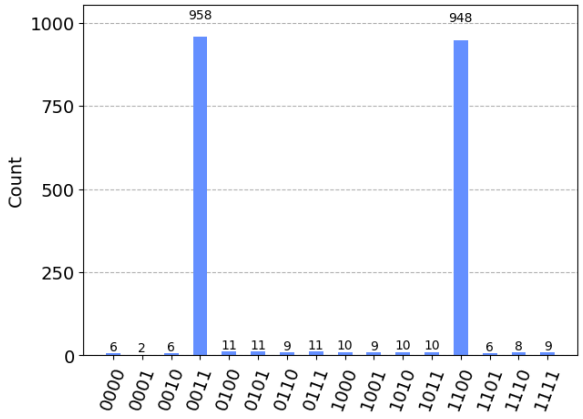


Figure 3: Results from qasm simulator

4 Conclusion

In this report, we learned how quantum walks are different from Classical Markov walks and how the power of superposition can provide a quadratic speed up to the walk search problem in the Quantum Walk Algorithm. It must be noted that there is nothing random in this algorithm thus the name of the algorithm is quite misleading whereas on the other hand, it helps us in drawing an analogy with the classical random walk. Here the walker is not moving at one node only but it can be thought of as taking steps in all directions simultaneously or put in other words the walker is in a superposition over all neighboring nodes. The true beauty of the algorithm lies in the heart of the Quantum Fourier transform and Phase Estimation. Through this process, we can find the distribution which does not change any further if the walker takes further steps i.e. state $|U\rangle$. Applying this to an example of a 4D hypercube, we efficiently found marked vertices.

References

[1] Ronald de Wolf. Quantum computing: Lecture notes, 2023.

[2] Abhijith J., Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Alexander Malyzhenkov, David Mascarenas, Susan Mniszewski, Balu Nadiga, Daniel O'malley, Diane Oyen, Scott Pakin, Lakshman Prasad, Randy Roberts, Phillip Romero, Nandakishore Santhi, Nikolai Sinitsyn, Pieter J. Swart, James G. Wendelberger, Boram Yoon, Richard Zamora, Wei Zhu, Stephan Eidenbenz, Andreas Bärtschi, Patrick J. Coles, Marc Vuffray, and Andrey Y. Lokhov. Quantum algorithm implementations for beginners. *ACM Transactions on Quantum Computing*, 3(4):1–92, jul 2022.

[3] R. Portugal. *Quantum Walks and Search Algorithms*. Quantum Science and Technology. Springer New York, 2013.

[4] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.