Derek Bowdle
Quantum Computation
EN.605. 728.81
Fall 2022

Quantum Random Walks: First Hitting Time

Preface

The purpose of this paper is show how first hitting times can be calculated using a Quantum Walk. To do this and understanding of random walks must first be built, what they are, how to represent them, how hitting times can be generated for classical random walks. Then a notion of how a quantum computer can be used to model a random walk will be established, followed by an exploration of what is ideal to model using a quantum computer and the possible advantages over its classical implementation. Finally, first hitting times using a quantum random walk will be explored.

What is a random walk

A random walk is so fundamental that it is difficult to describe without putting it into a formal mathematical description. It is one of the building blocks for stochastic processes and as such it is literally a starting point for many forms of analysis. We can dissect a informal meaning by breaking apart the component words. First the word to look at "walk" refers to a progression through space and time. While a random walk is always moving though or around some space there is a lot of variation to this. As we will see later, it could be moving around a line, a graph, or perhaps some n-dimensional plane. A random walk can also move through time in a discrete or continuous manner though the two are strongly connected and we can often model continuous processes with discrete time walks. The other word is "random" which describes how the "walk" or progression through space and time occurs. This greatly expands what can be explored with random walks because there are many ways to move randomly. In fact, we can describe the randomness of the movement with any probability function we like, or in the case of a Markov chain a function of the location itself. If you were to create a random walk where you move through some volume with movements described using a normal distribution you could model the movement of a gas molecule, also known as Brownian motion.[1] As we will explore later if the movement is described with a coin flip over a one-dimensional line, we can also model the classic gamblers ruin problem.

First, we establish a notion of a simple random walk on a one-dimensional line. To do this we start at some initial position $z \in \mathbb{Z}$ then incrementally take steps $X_1, X_2, \ldots, X_n$ each in either direction, depending on the outcome of a coin flip. We can define this a by defining n independent random variables $Y_1, Y_2, \ldots, Y_n$ such that $Y \sim \text{Bernoulli}(0.5)$ and define $X = 2Y - 1$ also known as a Rademacher distribution then our random walk is described as our series S_n

$$S_N = z + X_1 + X_2 + \cdots + X_n$$

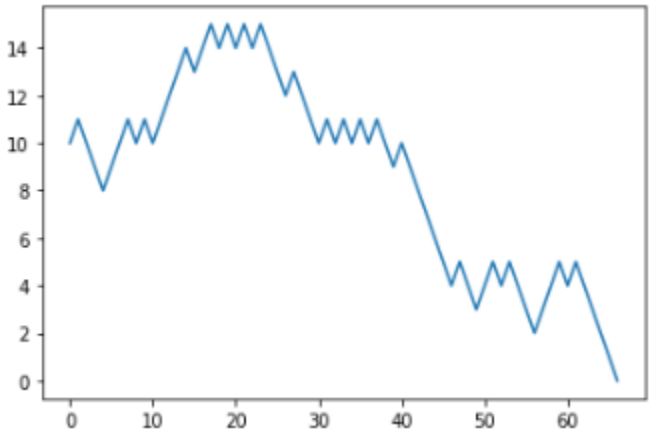
Or

$$S_n = S_{n-1} + X_n, n \geq 1 \text{ and } S_0 = z^2$$

[1] https://web.mit.edu/8.334/www/grades/projects/projects17/OscarMickelin/brownian.html
[2] https://www.math.ucla.edu/~biskup/PDFs/PCMI/PCMI-notes-1

Gamblers ruin

Let's walk through an example of a use case and implementation of this one-dimensional discrete time random walk. Let's say Tom has 10 dollars in his pocket and wants to try his luck at roulette. His strategy is to always bet one dollar on red and the casino he is playing at offers excellent odds, they don't have the 0 and 00 squares making his bet a true 50-50 odds, he is expected to win exactly half of the time. If he is set on playing 100 rounds what would a potential outcome of his decisions look like? We can explore this by generating a random walk using the inputs we described. Z=10



As you can see in this outcome Tom has lost all his money before reaching his 70th round in this specific outcome but this is just one of many possible outcomes, in fact for this scenario there are somewhere around $\sim 2^{100}$ possible outcomes or possible random walks that tom could take (it would be exactly 2^{100} if tom had access to more money if he ran out).

Hitting times

What if we want to analyze the cases in which Tom runs out of money and answer questions such as how often he will run out of money, when is the earliest he could run out of money, or on average if and when he is expected to run out of money, we would evaluate the hitting time of 0 where $S_n = 0$. To achieve these results through simulation we would generate some large number of random walks and calculate them using the sample to approximate the results. Let's say that each sample random walk $S_{n,i} \in S_{n,1}, \ldots, S_{n,m}$ for our sample of m random walks has a final term $S_{k,i}$ such that $S_{k,i} = 0$ or $S_{k,i} \in \mathbb{N}, k = 100$ then we can calculate the following:

How often will he run out of money:

$$\Pr(S_{k,i} = 0)$$

The earliest he could run out of money:

$$\min(k)$$

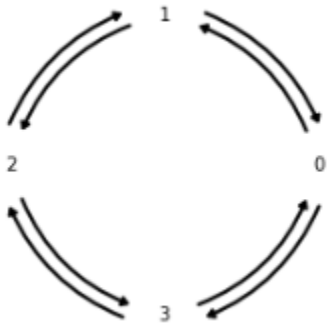
when he is expected to run out of money:

$$\frac{\sum k_i}{m}$$

The question about when he is expected to run out of money in our case is an invalid one. While many random walks reach 0 many do not, and if we were to unbind the time constraint *n*, we would see that many of the final terms become infinite showing us that while there is a chance of him reaching 0 he is not guaranteed to. For this reason, we will be focusing on analyzing the first two questions how often given a time constraint and the approximate earliest if could happen.

Random Walks on a graph

Let's now apply a random walk to a graph. For simplicity's sake the graph will have four nodes and equal probability edges. Each node is a position, and each edge is a possible movement to the next step. For a Markov chain edges can connect to any node including its self with any probability as long as the total probably of all the edges leaving a node add up to 1 or the node has no edges (which is essentially the same as only having an edge that loops back to itself)



Transition matrices

The most useful representation of this graph is its corresponding transition matrix. For a graph with *n* nodes, we can construct an *n* by *n* transition matrix A by setting $A_{i,j} = Pr(i\ to\ j)$. Therefore, the transition matrix for the graph show above would be:

$$\mathrm{A} = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix}$$

Hitting time for Random Walks on a graph

We can develop a notion for hitting times on a graph just as we did for the one-dimensional random walk. If we start at node 2 lets find the probability that in two steps we move to node 0. This can be done by taking the initial starting position as a vector:

$$\mathrm{x} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

and applying the transition matrix to it for every step we take:

$$\text{one step } = \text{Ax} = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.5 \\ 0 \\ 0.5 \end{pmatrix}$$

$$two\ steps\ = AAx = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0.5 \\ 0 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \\ 0 \end{pmatrix}$$

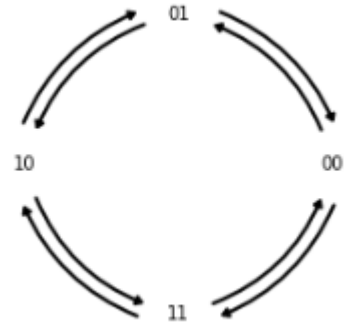
The resulting vectors tell us several things. The first step vector has no probability for node 0 and the second step vector has 0.5 probability of hitting node 0 therefore there is a 50% chance of hitting node 0 from node 2 after two steps and that this is also our first hitting time.

What is a quantum walk

A quantum walk or quantum random walk is an random walk simulated on a quantum computer. It is implemented in a very different way from most other classical random walks but also has some similarities to performing random walks on graphs. It functions through two components: nodes and coins. The first component is simple, we assign qubit states to nodes. If we have two qubits we can have four possible states (00,01,10,11) therefore we can assign each nodes a state:

$$0 \rightarrow |00\rangle, \qquad 1 \rightarrow |01\rangle, \qquad 2 \rightarrow |10\rangle, \qquad 3 \rightarrow |11\rangle$$

Each qubit has two possible states and when we combine *n* qubits we get 2^n possible states therefore if need *k* nodes we need $2^n = k \Rightarrow n = log_2(k)$ qubits to represent them. Applying our new states to the nodes would get us something like this



From there we can describe the transitions between the matrices. To do this we use what's called a Hadamard coin. Some refer to this as a quantum coin flip, but essentially it allows us to split our circuit into many possible outcomes while following very specific rules. It works by assigning additional qubits in the circuit and using a Hadamard gate to put the coin qubit in an undetermined state. From there we can perform an operation on our node qubits while controlling it using the coin qubit. We can then flip the coin qubit and perform another operation on the node qubits. A single Hadamard coin qubit allows

two operations, which could also be seen as edges, but just like with the nodes additional qubits double the number of possible operations or edges.

Creating a Quantum Walk

If we wanted to reconstruct the graph that we found the transition matrix for we would need one qubit for the coin so that we can travel both ways around the ring and two qubits for the four states. Next, we need to define operations that in the way we desire. In the problem above we can move in two directions, one increases resetting to 0 when we get to 4 and the other decreasing resetting to 3 when we pass 0. This can be done in a couple of ways but the idea is that we need a series of gates that provide the proper mapping that are also connected to the Hadamard coin. The mappings we would like to find gates for are:

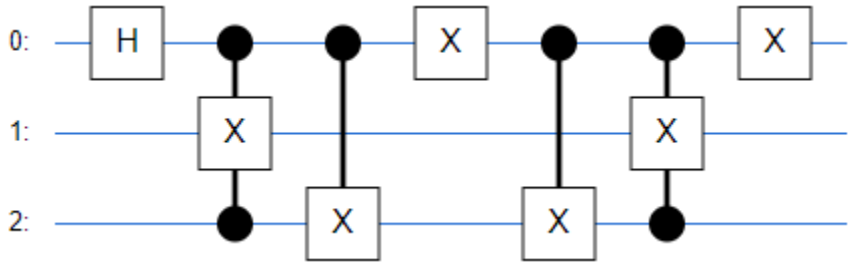
Increase mapping

$q1_i$	$q2_i$	$q1_f$	$q2_f$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Decrease mapping

$q1_i$	$q2_i$	$q1_f$	$q2_f$
0	0	1	1
0	1	0	0
1	0	0	1
1	1	1	0

This can be achieved by some controlled x gates, which flip the gate when the coin bit is set to 1. Essentially, we always flip the lower bit then we flip the next bit depending on what we flipped the first bit to or what the first bit was. Next, we need initialize the Hadamard coin and sandwich the gates that perform the mapping between the two x gates to flip the state of the Hadamard coin between operations. If q_0 is the Hadamard coin and q_1 is the highest order node qubit then the circuit for the



Then to simulate our quantum walk we would simply start with some input for node bits that tell us where we start, run it through the circuit above one time for every step. As I demonstrated using cirq in section 2.2 of the programming document we obtain the same first hitting time for the graph as above.

Advantages and Disadvantages

The largest advantage to the quantum random walk is that the number of nodes increases exponentially with the number of qubits we use to represent them. Let's say for the matrix above you don't keep track of probabilities and instead represent it with a, adjacency matrix (ones in place of probability values). You could still represent the graph and investigate how quickly we move through the graph given some initial position. For this representation you would need a bit for every value in the matrix and if we have n nodes then we would need n^2 bits to represent it. This is contrasted with the $log_2(n)$ qubits need for

all the nodes for the equivalent quantum walk. The difference is staggering when taken to the extreme; lets say you for some reason need to do a quantum walk on a graph that has a node for every atom on earth. With the classical representation this would be impossible, but because there are somewhere around 1.3 * 10^50 atoms on earth[3], with a quantum computer you would only need somewhere around 170 bits.

While the quantum advantage is real, it is obviously faced with many issues. The first is that we simply don't have error proof quantum systems yet, so until we do any advantage is not practical to implement. The next issue is with this is the symmetry that is required within the graphs for them to have a quantum circuit that corresponds to the way that the edges connect. I spent some time playing with different gate operations and everything they appear to be cyclical in some way. This can of course be useful for various applications but the set of graphs that a quantum walk is possible on is far smaller than the set of graphs you create a Markov chain for. Lastly there are interference effects that disturb the probabilities of many implementations in section 2.4 of the programming document you can easily see that distribution is not symmetrical like we would expect. This is predictable yet a nuisance to deal with when trying to compute certain values related to your random walk. In addition, when you run a quantum walk with a large time scale you see affects that often completely differ from what you are actually trying to represent.

[3] https://sciencenotes.org/how-many-atoms-are-in-the-world/#:~:text=in%20the%20World-,The%20number%20of%20atoms%20in%20the,around%201.3%20x%201050.&text=The%20number%20is%20an%20approximation,of%20atoms%20is%20always%20changing.