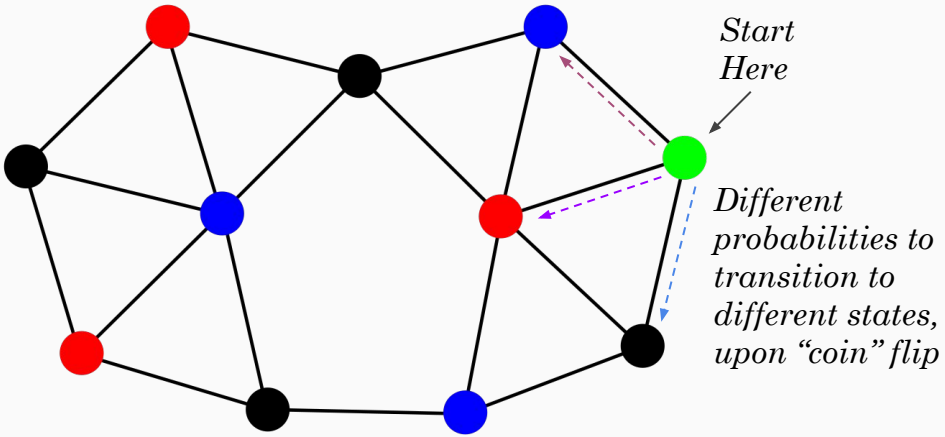
Exploring the Quantum Random Walk Search Algorithm

Davis Brown, Nick Konz, Eric Yelton

Classical Random Walks (RWs) on Graphs

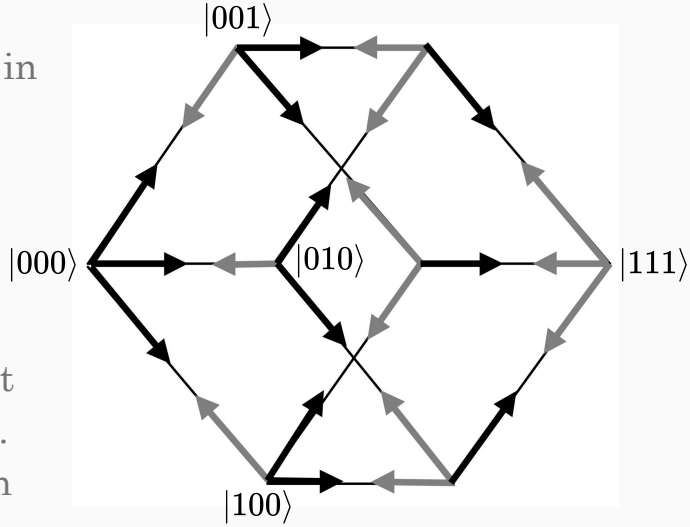


- Begin "walker" on some initial node, and then evolve it to other nodes with certain transition probabilities.
- Can *search* for a node in particular through this RW evolution.
  - Widely used, but worst case can take *O(N)* evaluations.
- How can this method be brought to the quantum realm?

Credit:
https://upload.wikimedia.org/wikipedia/commons/f/fc/Penny_graph_11_nodes.svg

Quantum Random Walks (QRWs)

- Imagine the walker transitioning to multiple nodes at once, in a *superposition*.
- Just as we flip a "coin" to decide which specific state to transition to/direction to take in a CRW, we can introduce a *quantum coin* that creates a superposition of multiple directions at once!
- This way, the *quantum* walker can explore multiple nodes at once, now giving a worst-case of $O(\sqrt{N})$ (oracle) evaluations.
- We take the approach of Shenvi et. al 2003, where our graph is an n-dimensional hypercube which has nodes that can be labeled by n qubits, each with a bit string $\vec{x}$ (see right).



Credit: Shenvi et. al 2003

Note: this is a *discrete time-step* QRW; there are also QRWs that evolve *continuously* in time.

How many qubits are needed?

- The quantum coin is characterized by the directions that the walker can take; since our graph is an n-cube, there are n distinct directions. So, you'll need $\log_2 n$ *coin* qubits to specify the direction(s) of the walker.
  - So, searching for some bit string of length n requires:
    i. n qubits to specify the position/node of the walker on the n-cube
    ii. $\log_2 n$ qubits to specify the state of the coin/specify the direction of the walker
- As such, each step of the QRW can be thought of as a single application of a unitary operator U to the state of the system, acting on a Hilbert space $\mathcal{H}^n \oplus \mathcal{H}^{2^n}$.
- Now, what is this operator U, exactly?

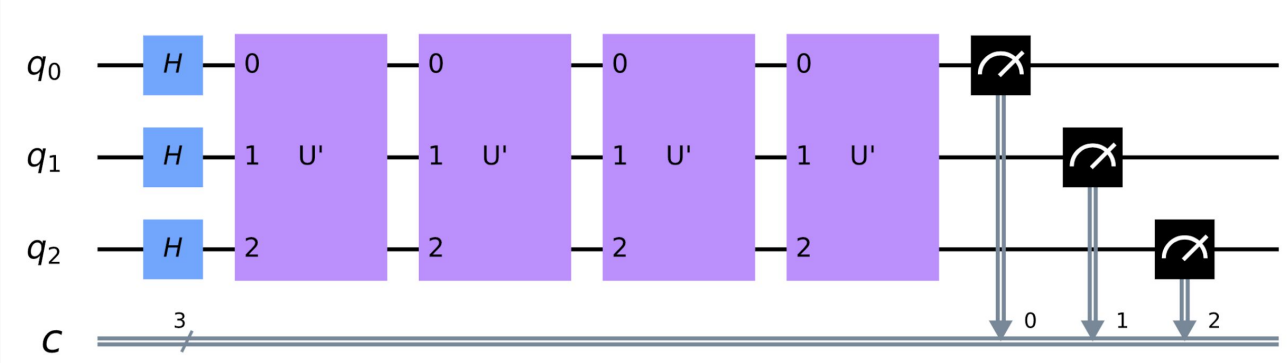
The Evolution Operator

- From Shenvi et. al 2008, we can write this unitary evolution operator as $U = SC$, where:
  a. S is the *shift* operator/matrix, that performs a controlled shift on the node/*n-cube* qubits according to the state of the quantum coin.
  b. C is the *coin* operator/matrix which flips the quantum coin.

- One more modification: so far, this quantum random walk isn't *searching*:
  - We need to mark our target state within U, perturbing $C \rightarrow C'$ so that then U becomes some U'.

Implementation with Qiskit

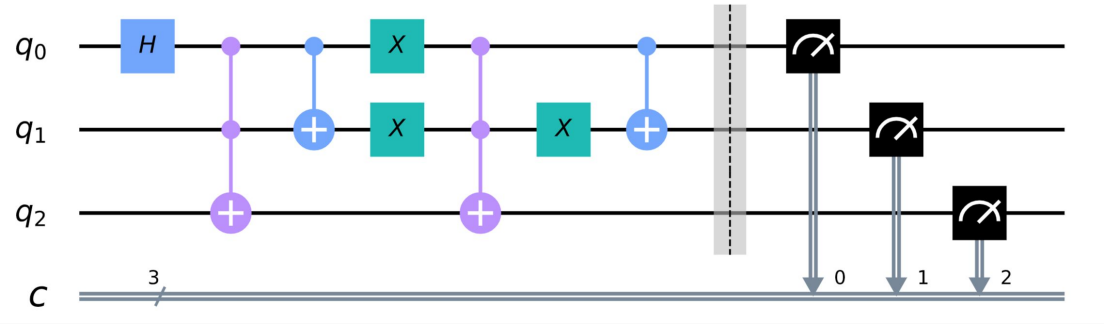
- We use Qiskit to write out these operators, the circuit, and the simulation (explicitly defined in the github repository under qrw/qrw.py).
- We initially define the matrices for the respective operators using NumPy, and then use the Operator class in Qiskit to define the (unitary) matrices as operators. We add the gates to the circuit using the unitary method.
- The circuit (from random_walk_search.ipynb in the repo) for $n = 2$ is below; however we implemented the code for arbitrary $N = 2^n$. We include the matrix defn. of U' to the right.

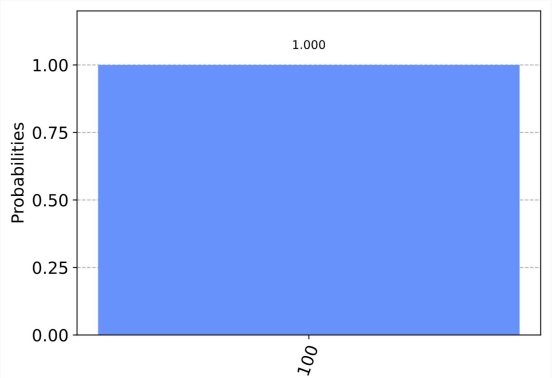



```
Matrix for U':
[[ 0.  0.  0.  0.  0.  1.  0.  0.]
 [-1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.]]
```

Simplification and the state histogram

- We use a tool to decompose the unitary evolution operator into quantum gates. We used a tool which largely used methods from a paper by Li, Roberts, and Yin.
- The original decomposition can be found in random_walk_search.ipynb. We argue that with a slight change in the shift operator (namely, making it flip the state of the coin as well as shift the direction vector), we can achieve the more compact circuit implementation below (shown with only one operator step for brevity). The corresponding state (with 4 steps) from the Aer simulator is also shown on the right.





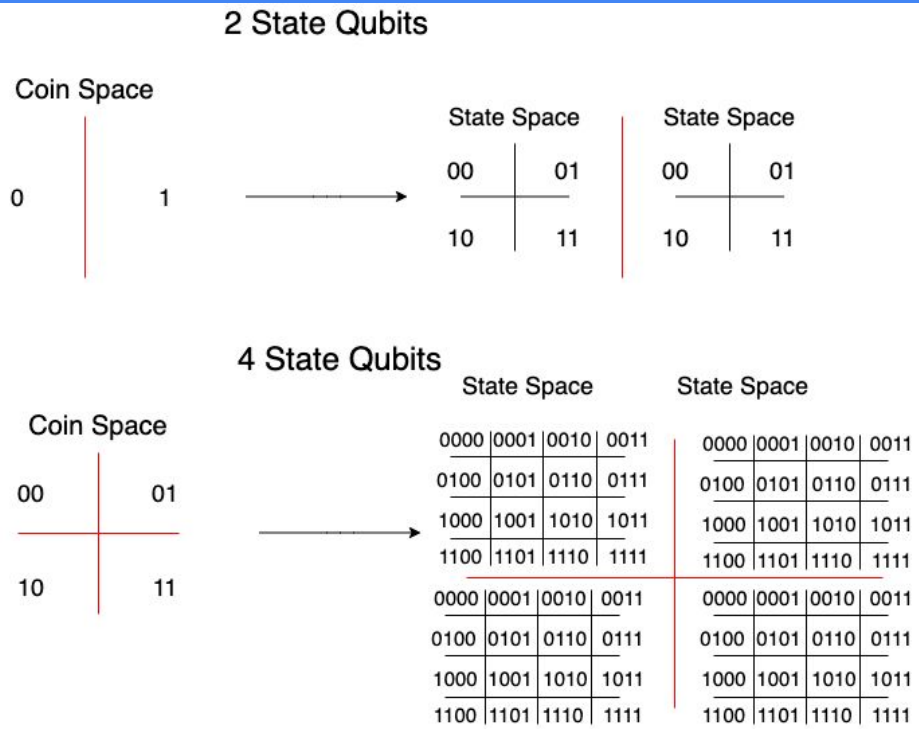
QRW Automata

Going beyond the search algorithm we can use Quantum Random Walks for other applications. One example we implemented was mapping both the coin and state space to a 2D grid of squares. The result from a QRW circuit is executed and measured a prescribed number of times. The resulting measurement is visualized by changing the color of the corresponding grid space.

Inspired by Cellular Automata and Conway's Game of Life - This code displays the the sampling of the state space and coin space by an arbitrary QRW circuit

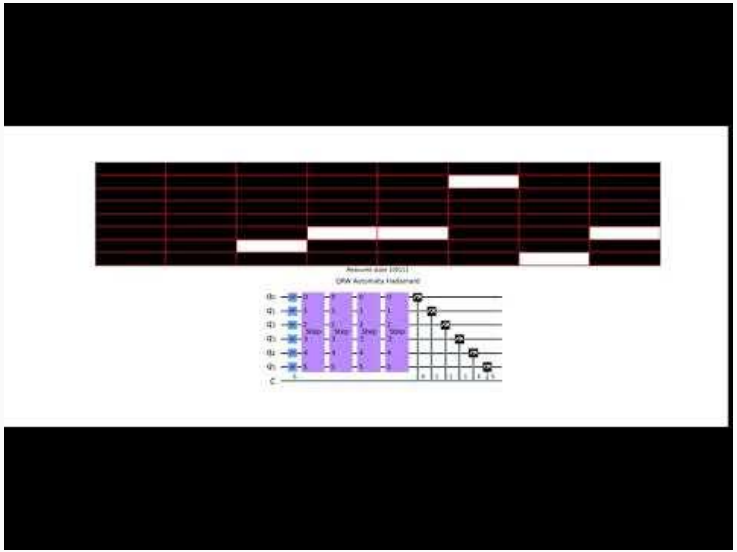
The Mapping

The coin space state is ‘mapped’ to the grid by dividing the grid into large regions. The large regions are further subdivided by the resulting state space.



Resulting QRW ‘Path’ Animation

Resulting animation for 4 state qubits with a Hadamard coin operator



Resulting animation for 4 state qubits with a Random Unitary coin operator