Quantum Rubik's Cube Solver

Alex Liu, Anthony Ou, Howard Zhong, Zachary Chin

January 2021

1 Abstract

In this project, we tackled the challenge of solving a simpler variant of a Rubik's cube through representing it as a Markov chain and using a quantum search algorithm. This allowed us to obtain a speedup over any classical algorithm by taking advantage of quantum superposition and parallel computation, particularly in the case of sequences with a large number of moves. We would also be able to extend this work to larger and more complex problems given the requisite hardware while maintaining the computational advantages provided by quantum computing.

2 Introduction

The Rubik's cube is a famous 3D combination puzzle in which a user sequentially twists various faces of the cube to obtain the original solved state from a scrambled state. Because there are over 43 quintillion potential permutations of the standard Rubik's cube, finding a sequence of moves which takes a scrambled state to a solved state can be computationally quite difficult. We are particularly interested in the problem of finding a move sequence with a given length which solves the cube. A classical algorithm brute-force searching through all of the possible move sequences of a given length would have to evaluate $O(N)$ sequences, where N is the total number of move sequences with that length. In contrast, a quantum search algorithm based on Grover's algorithm could complete the same search in $O(\sqrt{N})$ evaluations, which is potentially considerably faster. In this project, we aim to implement a quantum algorithm which solves a simplified Rubik's cube using a given number of moves. We hope this project may serve as a proof-of-concept prototype for a more advanced algorithm which might ultimately solve the classic $3 \times 3 \times 3$ Rubik's cube once quantum hardware becomes sufficiently computationally powerful.

3 Algorithm

Here we aim to find a specific sequence of a given length that will solve a simplified $2 \times 2 \times 0$ Rubik's cube, with one face being white and the other black. We specifically chose this cube because we were constrained to only having 11 qubits for the hardware, and this was among the most complex problems that we could feasibly solve.

The $2 \times 2 \times 0$ Rubik's cube (which we refer to as just a Rubik's cube for the remainder of this writeup) has four possible moves, but due to symmetry this can be simplified to just two, since rotating the left side is equivalent to rotating the right side (and rearranging the cube orientation). Hence, we can simplify our state space by fixing the top left corner to a given value and considering only the two moves of rotating the right two squares and rotating the bottom two squares.

Given that the top left corner is fixed (and without loss of generality, we can assume that it is black), there are only three possible states that the cube can be in. They are shown in Figure 1 below:

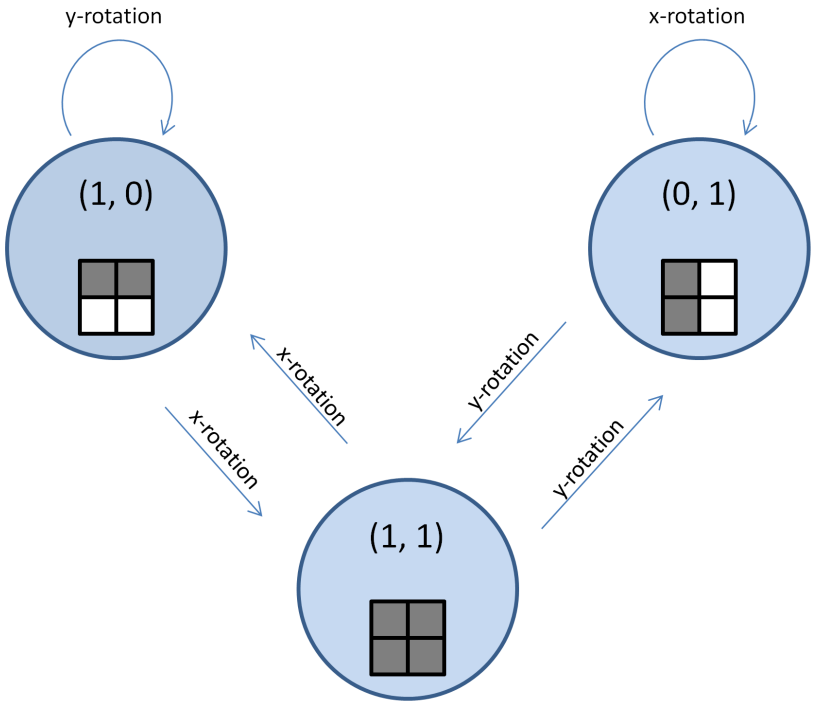

Figure 1: Markov Chain representing the states of the Rubik's cube and their potential transitions

We denote the upper right corner, bottom left corner, and bottom right corner by X, Y, and Z respectively, and define c_x as Y == Z and c_y as X == Z. Note that the three states from left to right correspond to the (c_x, c_y) pairs (1, 0), (1, 1), and (0, 1), and because c_x and c_y are defined in a symmetric manner to 0 and 1, these hold similarly if the upper left corner is white. We denote our Rubik's cube "state" as (c_x, c_y), for which the reasoning will become clear soon.

First, we note that (1, 1) is the solved state of the cube. Now we consider making either of the possible moves on the cube from a given state - define the "x rotation" by rotating the bottom two squares of the cube, and the "y rotation" by rotation the right two squares of the cube. Note that the x rotation maps (Y, Z) to (NOT(Z), NOT(Y)) since the cubes flip rotation and orientation, and as a result c_x stays invariant. If (Y, Z) is (0, 1) or (1, 0), the x rotation doesn't change the state of the cube, so c_y also remains the same. Otherwise, the rotation will flip the Z value, which will change c_y. Hence, the overall result of this operation is that c_x remains constant and c_y flips if c_x is 1 - this is equal to a CNOT gate with control c_x and target c_y. Similarly, the y rotation is equivalent to a CNOT gate with control c_y and target c_x.

Using these nice properties of c_x and c_y, we are then able to construct a quantum circuit that uses c_x and c_y as qubits and can perform moves (denoted by m_i) on these qubits. Specifically, we can denote the x rotation by the 0 move and the y rotation by the 1 move. This lets us construct our circuit by using a Toffoli gate with controls NOT(m_i) and c_x and target c_y to represent the x rotation and similarly using a Toffoli gate with controls m_i

and c_y and target c_x to represent the y rotation.

Now that we are able to apply moves to our quantum state, we can implement Grover's search algorithm[1] to figure out a solution. Grover's search algorithm works by using an oracle to flip the phase of the correct solution and a diffuser to reflect all the states across the mean, increasing the amplitude of the correct state. Repeating this a specific number of times increases the amplitude of the correct state enough that a majority of measurements will yield that value. The exact number of times that the oracle/diffuser need to be applied depends on the number of states.

To implement the oracle, we first initialize n qubits to the 0 state (where n is the given length of the sequence) and apply the Hadamard operator on all of them to get a superposition of all possible states. We then apply the moves to our state qubits (which are initialized to the initial cube state) and then add another Toffoli gate using the state qubits as controls and an output qubit as a target. Since (1, 1) is the solved state, the output qubit only flips for a solved cube. We initialize the output qubit to the minus state, or $|0\rangle - |1\rangle$, so that the X operator causes it to change to $|1\rangle - |0\rangle$, which is effectively a phase inversion, as desired. We then invert the first part of the circuit to return the circuit to the initial value, completing the oracle. The full quantum circuit for the oracle (with two moves) can be seen in Figure 2 below.

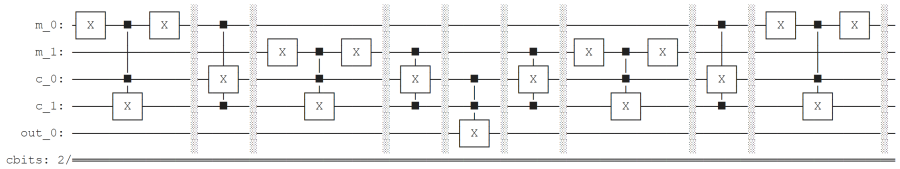

Figure 2: Grover Oracle with length two sequences

For the diffuser, we cited the implementation from qiskit's official website for arbitrary-length diffusers[2].

We note that the Markov chain described above in Figure (X) has a stationary distribution with $\frac{1}{3}$ probability at each node. With n qubits representing the move sequence and n sufficiently large, the probability of a given sequence solving the cube approaches $\frac{1}{3}$. This means that, regardless of the starting state, the N used in Grover's algorithm is approximately 3, so the angle theta is approximately 70°. This ensures that 1 pass of Grover's algorithm will give roughly a 92% chance of returning a correct solution, which is remarkably high. In our case, any n greater than or equal to 3 is sufficiently large.

A simulation with $n = 4$, shown in Figures 3 and 4 below, confirms that we are able to find a correct solution of length 4 with a very high probability, as expected.

[1] https://qiskit.org/textbook/ch-algorithms/grover.html
[2] https://qiskit.org/textbook/ch-algorithms/grover.html

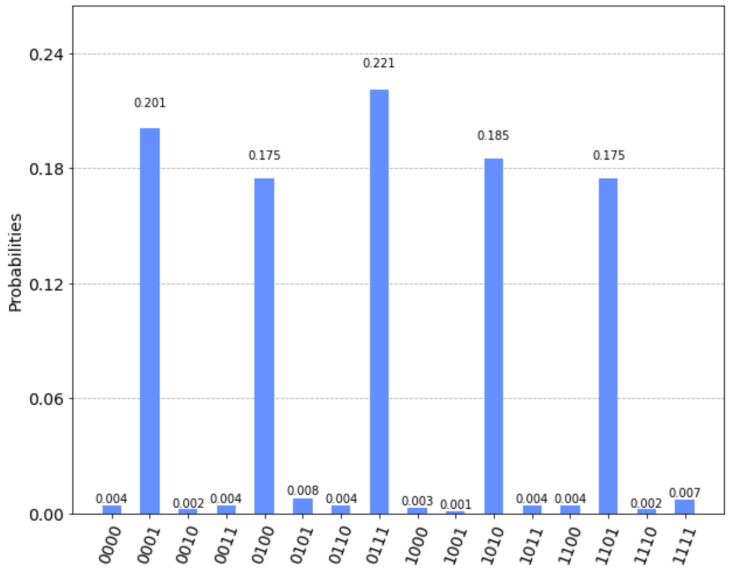

Figure 3: $n = 4$ simulation of Grover's algorithm starting from (1, 0) state with 1000 shots

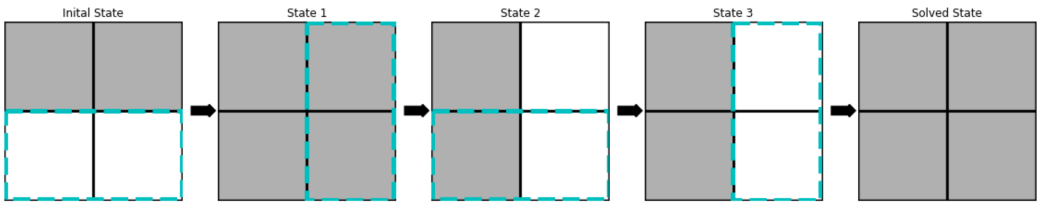

Figure 4: Solution sequence '1010' from Figure 3 (note that qiskit reverses the bit sequence)

4 Implementation

We implemented the algorithm with Python in Jupyter Notebooks using the qiskit module and will eventually run it on the IonQ's trapped ion quantum computer. All plots were made using MatPlotLib.

5 Conclusions

Through this project, we were able to successfully find a solution of arbitrary length for a $2 \times 2 \times 0$ Rubik's cube on a quantum computer using Grover's search algorithm. Although this problem is simple, this implementation is still faster than a classical algorithm to solve the same problem and as we progress to more challenging problems our quantum algorithm will become increasingly superior. This approach of using Grover's algorithm to solve a Rubik's cube could potentially be generalized to an arbitrary Markov state problem, including but not limited to larger Rubik's cubes. To do so would be challenging but could yield great utility.

6 Future work

Future work will focus on scalability. While Grover's search algorithm solves the simplified problem, as we consider larger cubes with larger problem spaces several problems present themselves: memory storage, oracle complexity and solution space ambiguity. As stated before, transitioning to a larger cube will have an exponential increase in the state space and will require more qubits to implement our algorithm. With a larger cube, the

oracle also becomes increasingly complicated to design efficiently, especially because we need to know the exact size of the solution space in order to be able to consistently obtain the correct solution; however, for the $3 \times 3 \times 3$ Rubik's cube this is ambiguous. For a slight variation of the problem where we seek to find a solution to the Rubik's cube in N steps, then for large N we can assume that all cube states are equally likely end states. The ratio of the solution space to the total space then becomes trivial to obtain. While these are significant obstacles to scalability, they are not intractable. With time and effort we believe that we can design an efficient algorithm that will run given enough working qubits.