Using Qbraid and Grover’s Algorithm to Create a Hamiltonian Cycle

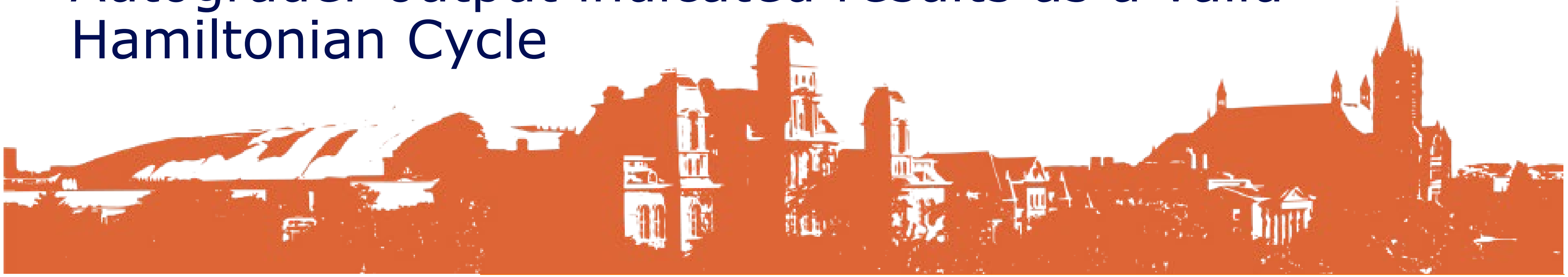
Submitted By

QuantumCuse Qbraid Team

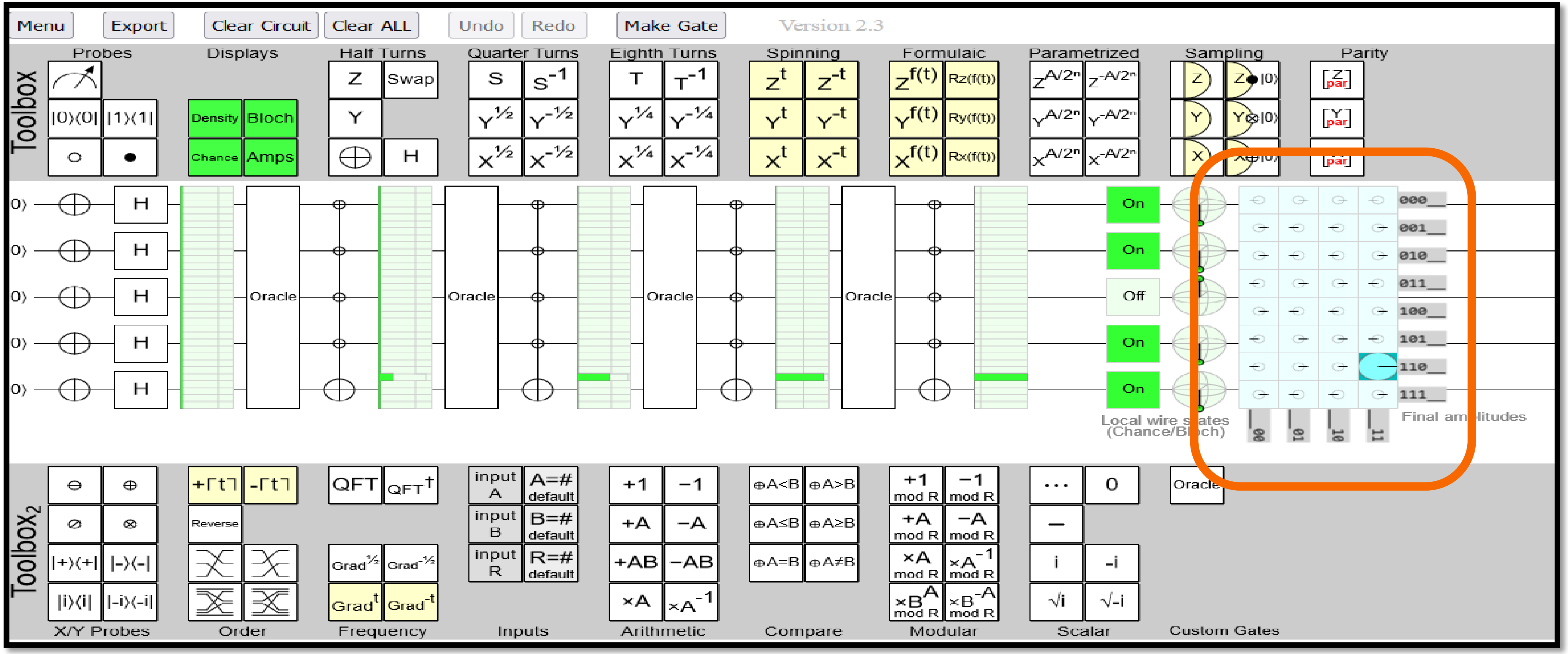
Vincent Plaza
David Ladd
Fundi Juriasi

Overview

- Qbraid Grover’s Algorithm
- Relied on code supplied by QC Hack for the Grover Algorithm
- Studied quantum computing simulator application – really cool!
- Expanded logic with python code to solve the problem
- Autograder output indicated results as a valid Hamiltonian Cycle



Quirk – Drag and Drop Quantum Simulator https://algassert.com



Methodology

Unitary Matrix 32x32

```
# YOUR CODE HERE#Need to create an n=32 unitary matrix!
# YOUR CODE HERE
@circuit.subroutine(register=True)
def ccz(targets=[0, 1, 2, 3, 4]):
    """
    implementation of three-qubit gate CCZ
    """
    # define three-qubit CCZ gate
    ccz_gate = np.array([[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
```

Oracle Dictionary – Derived From Simulator

```
[12]: # All possible items and their corresponding oracles
      # define oracle dictionary using this CCZ gate
      oracle_sim = {"00000": Circuit().x([0,1,2,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,2,3,4]),#0
                    "00001": Circuit().x([0,1,2,3]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,2,3]),#1
                    "00010": Circuit().x([0,1,2,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,2,4]),#2
                    "00011": Circuit().x([0,1,2]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,2]),#3
                    "00100": Circuit().x([0,1,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,3,4]),#4
                    "00101": Circuit().x([0,1,3]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,3]),#5
                    "00110": Circuit().x([0,1,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1,4]),#6
                    "00111": Circuit().x([0,1]).ccz(targets=[0, 1, 2, 3, 4]).x([0,1]),#7
                    "01000": Circuit().x([0,2,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,2,3,4]),#8
                    "01001": Circuit().x([0,2,3]).ccz(targets=[0, 1, 2, 3, 4]).x([0,2,3]),#9
                    "01010": Circuit().x([0,2,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,2,4]),#10
                    "01011": Circuit().x([0,2]).ccz(targets=[0, 1, 2, 3, 4]).x([0,2]),#11
                    "01100": Circuit().x([0,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,3,4]),#12
                    "01101": Circuit().x([0,3]).ccz(targets=[0, 1, 2, 3, 4]).x([0,3]),#13
                    "01110": Circuit().x([0,4]).ccz(targets=[0, 1, 2, 3, 4]).x([0,4]),#14
                    "01111": Circuit().x([0]).ccz(targets=[0, 1, 2, 3, 4]).x([0]),#15
                    "10000": Circuit().x([1,2,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([1,2,3,4]),#16
                    "10001": Circuit().x([1,2,3]).ccz(targets=[0, 1, 2, 3, 4]).x([1,2,3]),#17
                    "10010": Circuit().x([1,2,4]).ccz(targets=[0, 1, 2, 3, 4]).x([1,2,4]),#18
                    "10011": Circuit().x([1,2]).ccz(targets=[0, 1, 2, 3, 4]).x([1,2]),#19
                    "10100": Circuit().x([1,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([1,3,4]),#20
                    "10101": Circuit().x([1,3]).ccz(targets=[0, 1, 2, 3, 4]).x([1,3]),#21
                    "10110": Circuit().x([1,4]).ccz(targets=[0, 1, 2, 3, 4]).x([1,4]),#22
                    "10111": Circuit().x([1]).ccz(targets=[0, 1, 2, 3, 4]).x([1]),#23
                    "11000": Circuit().x([2,3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([2,3,4]),#24
                    "11001": Circuit().x([2,3]).ccz(targets=[0, 1, 2, 3, 4]).x([2,3]),#25
                    "11010": Circuit().x([2,4]).ccz(targets=[0, 1, 2, 3, 4]).x([2,4]),#26
                    "11011": Circuit().x([3]).ccz(targets=[0, 1, 2, 3, 4]).x([3]),#27
                    "11100": Circuit().x([3,4]).ccz(targets=[0, 1, 2, 3, 4]).x([3,4]),#28
                    "11101": Circuit().x([3]).ccz(targets=[0, 1, 2, 3, 4]).x([3]),#29
                    "11110": Circuit().x([4]).ccz(targets=[0, 1, 2, 3, 4]).x([4]),#30
                    "11111": Circuit().x([0]).ccz(targets=[0, 1, 2, 3, 4]).x([0]),#31
                   }
```

Results

```
# Print the solution
g2.hamCycle()

# This code is contributed by Divyanshu Mehta
```

```
Solution Exists: Following is one Hamiltonian Cycle
0 1 2 4 3 0

Solution does not exist

13]: False
```

Verify your solution ¶

Using our autograder, input the solution you and your teammates have come up with. Your solution should be a list of integers beginning with the initial node to traverse the graph and ending with the same node to complete the "cycle".

```
# feed your solution to the auto-grader in the following format:
from auto_grader import is_hamiltonian_cycle
path = [5,6,1,7,4,2,0,3,5]
is_hamiltonian_cycle(graph,path)
```

```
Success: path is a Hamiltonian cycle
True
```

Thank you.

See y’all next year!

QuantumCuse Qbraid Team:

Vincent Plaza – vmplaza@syr.edu
David Ladd – djladd@syr.edu
Fundi Juriasi – fjuriasi@syr.edu