QAOA_In_Silq_Report

March 4, 2023

[ ]:
```python
import os
import re
import subprocess

import numpy as np
import networkx as nx

import pandas as pd

from qiskit import *
from qiskit.visualization import plot_histogram, plot_state_city
```

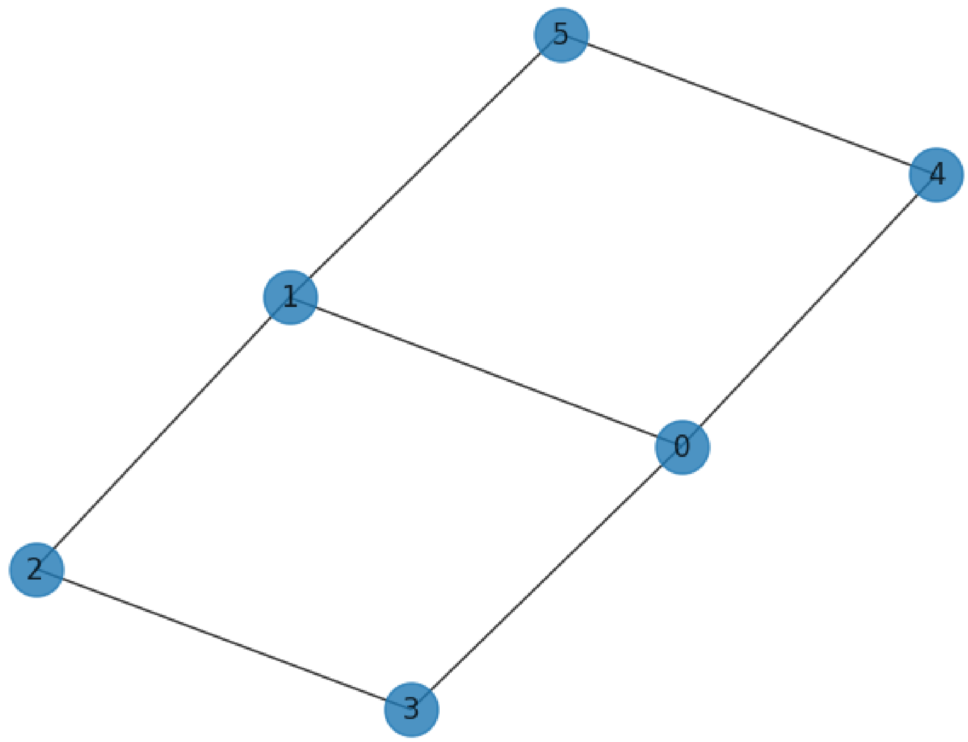
1 QAOA In Silq - Max Cut Graph

Author: Ryan Dougherty

UUID: 606085269

First, lets take a look at the 6 node graph I used in the program:

[ ]:
```python
G6 = nx.Graph()
G6.add_nodes_from([0, 1, 2, 3, 4, 5])
G6.add_edges_from([(0, 1), (1, 2), (2, 3), (3, 0), (4, 5), (0, 4), (5,1)])
nx.draw(G6, with_labels=True, alpha=0.8, node_size=500)
```



1.1 Silq Program and Results

I took the edge parameters from this graph and inputted them into my QAOA.slq file included in my report. Here's the full state and amplitude output from the 6 qubit state:

[ ]:
```
bashCommand = "silq --run ./QAOA.slq"
process = subprocess.Popen(bashCommand.split(), stdout=subprocess.PIPE)
output, error = process.communicate()

# Raw string outpput from Silq
state_sr = output.decode("utf-8", "strict")

# Regex setup to parse state vector into a dictionary
state_regex = re.compile(r'((?<=\().+?(?=\)))')
state_matches = re.findall(state_regex, state_sr)

state_dict = {}
for i in range(0, len(state_matches), 2):
    state_amp = complex(state_matches[i].replace('i', 'j'))
    state = state_matches[i+1].replace(',', '')
    state_dict[state] = np.abs(state_amp * state_amp.conjugate())
```

```python
# Porting state vector into DataFrame for cleaner output
state_df = pd.DataFrame(sorted(state_dict.items()), columns=['State',
 'Probability'])
pd.set_option('display.max_rows', state_df.shape[0]+1)
state_df
```

[ ]:

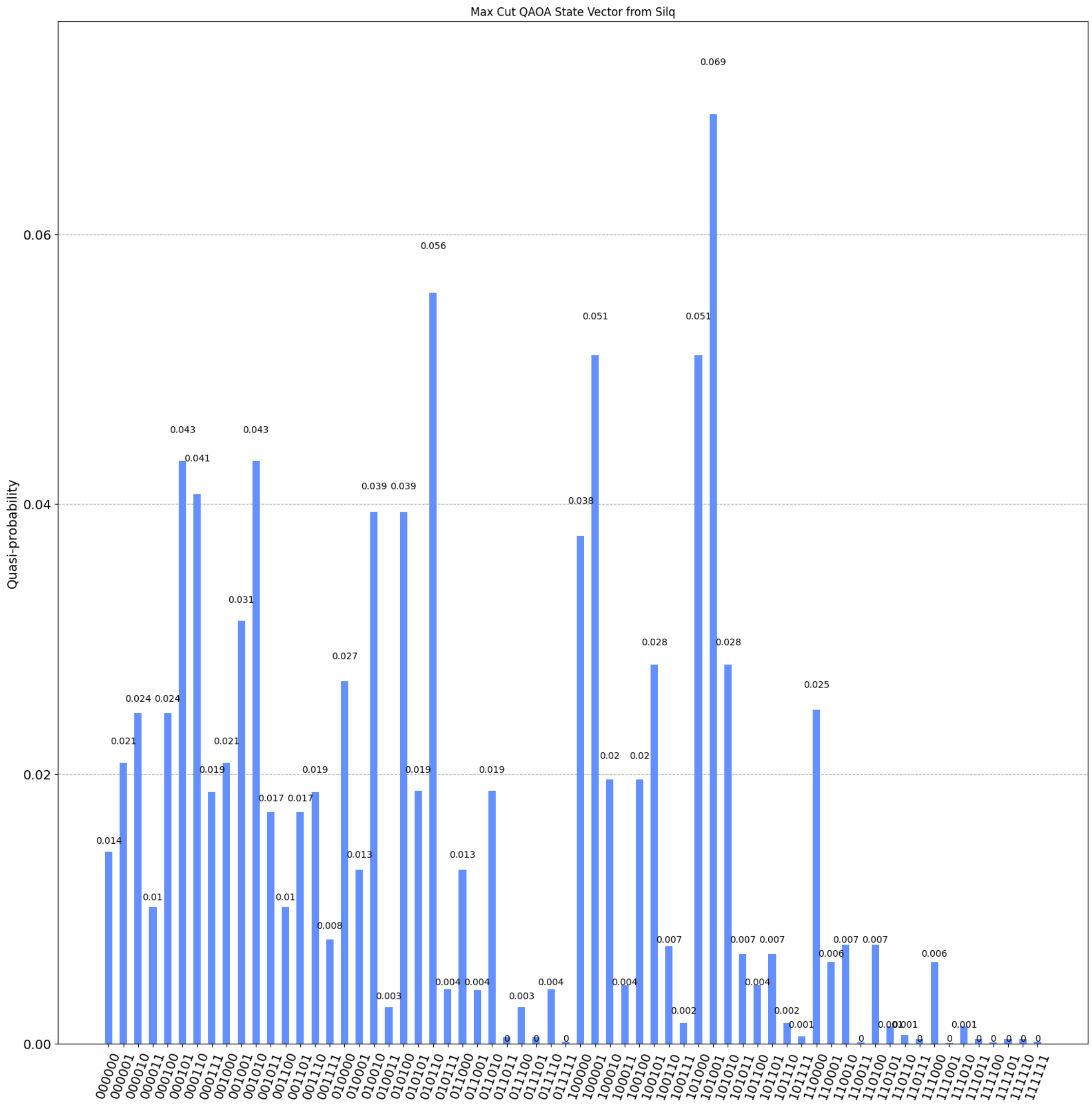
	State	Probability
0	000000	0.014239
1	000001	0.020837
2	000010	0.024500
3	000011	0.010122
4	000100	0.024500
5	000101	0.043207
6	000110	0.040775
7	000111	0.018665
8	001000	0.020837
9	001001	0.031371
10	001010	0.043207
11	001011	0.017165
12	001100	0.010122
13	001101	0.017165
14	001110	0.018665
15	001111	0.007753
16	010000	0.026865
17	010001	0.012913
18	010010	0.039436
19	010011	0.002725
20	010100	0.039436
21	010101	0.018780
22	010110	0.055686
23	010111	0.004057
24	011000	0.012913
25	011001	0.003999
26	011010	0.018780
27	011011	0.000477
28	011100	0.002725
29	011101	0.000477
30	011110	0.004057
31	011111	0.000159
32	100000	0.037647
33	100001	0.051036
34	100010	0.019594
35	100011	0.004319
36	100100	0.019594
37	100101	0.028131
38	100110	0.007213

```
39  100111     0.001537
40  101000     0.051036
41  101001     0.068916
42  101010     0.028131
43  101011     0.006633
44  101100     0.004319
45  101101     0.006633
46  101110     0.001537
47  101111     0.000540
48  110000     0.024739
49  110001     0.006076
50  110010     0.007333
51  110011     0.000095
52  110100     0.007333
53  110101     0.001283
54  110110     0.000652
55  110111     0.000367
56  111000     0.006076
57  111001     0.000040
58  111010     0.001283
59  111011     0.000351
60  111100     0.000095
61  111101     0.000351
62  111110     0.000367
63  111111     0.000132
```

Now, let's visualize these state probabilities on a histogram:

[ ]:
```
plot_histogram(state_dict, figsize=(20,20), title="Max Cut QAOA State Vector␣
↪from Silq")
```

[ ]:



1.2 Discussion

Based off these results for a 6 node graph, we can see the highest amplitudes for the strings '101001' and '010110' - where each 0 and 1 corresponds to the subdivided graph edges. This implies that we have two graphs cut to be 0,2,5 and 1,3,4 which matches up perfectly with our expected results for max cut.

Its worth noting that this algorithm can be easily tweaked. The graph edges and number of qubits can be edited in the `PARAMETERS` section at the top of the silq file. The beta and gamma parameters are a bit tricker, since they'd need to be optimized through another method. I simply chose the beta/gamma parameters optimized by the `scipy.optimize` function in my previous assignment. This step could easily be repeated for any additional parameterization.

1.3 Pros and Cons of Silq

1.3.1 Pros

Silq definitely does control operations quite well. I like the ability to do a controlled operation with a single if statement; It's a lot more readable than doing 'cnot' and the likes in qiskit. The methods silq uses to define quantum vs classical variables is quite nice. It really helps distinguish between what's my circuit and what's classical computation acting on my circuit.

1.3.2 Cons

I found that Silq lacks quite a bit of basic features you see in a lot of modern languages. It's inability to get array lengths or specify array sizes is a bit of an annoyance. As mentioned above the `if` statement syntax is nice, however, it doesn't allow you to test against non-constant circuit values. This means you need to swap a qubit with a constant ancilla register, which is kind of a headache.

1.4 Conclusion

All in all, I found this assignment quite fun. Its enjoyable to try out a quantum language and I look forward to seeing new updates and additions to Silq going forward.