QAOA for Max2SAT Report

Rishab Khurana

December 2023

1 Design

To parameterize in the number of qubits, I created a function called QAOA that took n (the number of qubits) as input, where n also represented the number of variables in the Max2SAT problem. Because the separator is dependent on the input expression for the Max2SAT problem, I black-boxed the separator (just like how we black-box function oracles for DJ, BV, Simon, and Grover). Essentially, I treat the separator as a representation of the boolean expression that is inputted – just like we treated U_f to be a representation of the function f to be inputted in DJ, BV, Simon, and Grover's.

2 Evaluation

In testing the program, I utilized Boolean expressions that had one optimal solution. For example, we can take $x_0 \cap (x_0 \cap x_1)$. From the lecture, we have seen that with $\gamma = \pi/4$ and $\beta = \pi/4$, we have that 11 has a 64% chance of being outputted. So to test this, I implemented the separator circuit for $x_0 \cap (x_0 \cap x_1)$ as

```
if (boolean_string == "x0 ^ (x0 ^ x1)"): #1-bit boolean expression (SAT)
    sep = QuantumCircuit(3, name="Seperator")
    sep.cp(-1*gamma, 0, 2)
    sep.mcp(-1*gamma, [0,1], 2)
    return sep
```

and ran QAOA with this separator and mixer using $\beta = \pi/4$ 5000 times and outputted the frequency of 11. Here is the code for the 5000 iterations:

```
# For testing purposes
n = 5000 #iterations of QAOA
all_outs = []
for i in range(n):
    #creating the seperator dependent on gamma
    sep = getSeperator(boolean_expression, gamma)

    #Applying QAOA for with beta
    circ, out = QAOA(sep, num_qubits, beta)

    all_outs.append(out)
print((all_outs.count('11'))/len(all_outs))
```

And here is the output:

```
(base) user@wifi-131-179-53-98 QAOA (HW 16) % python3 QAOA.py
0.636
```

which means the output is about on par with the results from the lecture. By using $\beta = 3\pi/4$, I tested for a different mixer and got the following result:

```
(base) user@wifi-131-179-53-98 QAOA (HW 16) % python3 QAOA.py
0.0404
```

which is *significantly* lower than the output of 0.636 from $\beta = \pi/4$.

As the number of qubits grows, the accuracy of the output seems to decrease (with the same mixers) – to increase the number of qubits I used the separator for the expression $(x0 \cap x1) \cap (x1 \cap x2)$ but with the same mixer. Here are the results (measure the frequency

of 111, which is optimal for this expression):

```
(base) user@wifi-131-179-53-98 QAOA (HW 16) % python3 QAOA.py
0.489
```

which is about 15% lower than with 2 qubits. With 4 qubits, we use the expression $(x_0 \cap x_1) \cap (x_1 \cap x_2)$ $cap(x_2 \cap x_3)$ to get about a 47% frequency for 1111, which is about the same as 3 qubits, but as soon as we get to 5 qubits with the expression $(x_0 \cap x_1) \cap (x_1 \cap x_2) \cap (x_2 \cap x_3) \cap (x_3 \cap x_4)$, we get an accuracy of around 27%, which is about 20% lower from the 4 qubits. Thus, we have the following diagram for these similar expressions (all with the same β):

$$1\ qubit \rightarrow 84\%$$

$$2\ qubit \rightarrow 64\%$$

$$3\ qubits \rightarrow 49\%$$

$$4\ qubits \rightarrow 47\%$$

$$5\ qubits \rightarrow 27\%$$

Thus for 5 qubits, a different β may be needed for optimal results. For execution times, I get the following diagram for mixer using $\beta = \pi/4$:

$$1\ qubit\ \rightarrow \approx 0.0190\ s$$

$$2\ qubit \rightarrow \approx 0.0192\ s$$

$$3\ qubits \rightarrow \approx 0.0383\ s$$

$$4\ qubits \rightarrow \approx 0.0496\ s$$

$$5\ qubits \rightarrow \approx 0.0521\ s$$

When introducing noise (using a noise simulator instead of a quantum computer due to queue times), we have the following accuracies

$$1\ qubit \rightarrow 67\%$$

$$2\ qubit \rightarrow 38\%$$

$$3\ qubits \rightarrow 31\%$$

$$4\ qubits \rightarrow 31\%$$

$$5\ qubits \rightarrow 16\%$$

with similar execution times. Therefore, the simulator has about 20% less accuracy in most cases (except the last one).

Using a different mixer drastically changed the accuracy. With $\beta = \pi/4$, we had about 64% accuracy with 2 qubits, but with $\beta = \pi/2$, we had about 40% accuracy. This trend was followed with more qubits. However, there seemed to be slightly faster execution times with $\beta = \pi/2$, but nothing too significant.

3 README

For a boolean expression B, one first needs to construct a separator circuit for B. Then, one needs to implement the circuit in qiskit in QAOA.py. Below is an example function that holds several separators given a string representing the boolean expression:

```
def getSeperator(boolean_string, gamma):
    if (boolean_string == "x0"):
        sep = QuantumCircuit(2)
        sep.cp(-1*gamma, 0, 1)
        return sep
    if (boolean_string == "(x0 ^ x1)"): #1-bit boolean expression
        sep = QuantumCircuit(3, name="Seperator")
        sep.mcp(-1*gamma, [0,1], 2)
        return sep
    if (boolean_string == "(x0 ^ x1) ^ (x1 ^ x2)"): #3-bit boolean expression
        sep = QuantumCircuit(4, name="Seperator")
        sep.mcp(-1*gamma, [1,2], 3)
        sep.mcp(-1*gamma, [0,1], 3)
        sep.mcp(-1*gamma, [0,1,2], 3)
        return sep
    if (boolean_string == "(x_0 ^ x_1) ^ (x1 ^ x2) ^ (x2 ^ x3)"):
        sep = QuantumCircuit(5, name="Seperator")
        sep.mcp(-1*gamma, [0,1], 4)
        sep.mcp(-1*gamma, [1,2], 4)
        sep.cp(-1*gamma, [2,3], 4)
        return sep
    if (boolean_string == "(x_0 ^ x_1) ^ (x_1 ^ x_2) ^ (x_2 ^ x_3) ^ (x_3 ^ x_4)"):
        sep = QuantumCircuit(6, name="Seperator")
        sep.mcp(-1*gamma, [0,1], 5)
        sep.mcp(-1*gamma, [1,2], 5)
        sep.mcp(-1*gamma, [2,3], 5)
        sep.mcp(-1*gamma, [3,4], 5)
        return sep
```

Once the seperator is implemented, one must choose the γ and β they want to use and call the getSeperator() function with γ and QAOA() with the Seperator, β, and n (the number of qubits, which is $1 + |\{vars.\ in\ boolean\ expression\}|$. Below is an example call of QAOA() with the boolean expression $(x_0 \cap x_1) \cap (x_{12}) \cap (x_2 \cap x_3) \cap (x_3 \cap x_4)$ (which has 5 qubits)

```
gamma = math.pi/4
beta = math.pi/4
boolean_expression = "(x_0 ^ x_1) ^ (x_1 ^ x_2) ^ (x_2 ^ x_3) ^ (x_3 ^ x_4)"
num_qubits = 5 #number of qubits/the number of variables in the boolean expression
noise = False
#output for mixer with pi/4
beta = math.pi/4
start_time = time.time()
sep = getSeperator(boolean_expression, gamma)
circ,out = QAOA(sep, num_qubits, beta, noise)
print("Time taken is", time.time() - start_time, end="")
print(", and the Output with mixer using beta = pi/4: ", out)
```

The user may also decide whether or not they want noise by changing the noise boolean. Once all the parameters are tuned, the use must go to their console and call the script in terminal by running the command

python3 QAOA.py

Then, the desired output should be there.