QAOA Supporting material

Group Eve: Yuyang Qin,Tingxu Ren,Ke Chen

In this project, we fucus on the optimization and simulation of QAOA algorithm and its implementation on the Max-Cut problem in Python.

First we implement the QAOA utilizing unitary matrix operator in numpy form. Then we use grid search, Bayesian Optimization and L-BFGS-B to find the best β, γ. With the best β, γ,we construct its quantum circuits and simulate them both with and without noise using IBM qiskit SDK.

Data Preparation

`graphic_in.py` : To prepare a random graph with n nodes and get the Max-Cut classic result of it, where `generateGraph()` could new a graph and `graphic(n,edge.ask_min())` provides its Max-Cut answer as well as division scheme in hexadecimal. The graph will be save into `grapy_in.npy`

`graphic_print.py`: draw the graph using networkx SDK.

Numpy Simulation and Parameter Optimization

In QAOA algorithm, we have

$$|\vec{\gamma},\vec{\beta}\rangle = U(B,\beta_p)U(C,\gamma_p)\dots U(B,\beta_1)U(C,\gamma_1)|s\rangle$$

where

$$U(C,\gamma) = e^{-i\gamma C}, U(B,\beta) = e^{-i\beta B}$$

which could be achieved by `scipy.linalg import expm`.

In our experiment, utilizing $e^{-i\gamma C} = Ve^{-i\gamma\Lambda}V^{-1}$ will lead to precision disasters. We even found that $e^{-i\gamma C}$ obtained in this method isn't an unitary matrix anymore.

Now, given an array of $|\vec{\gamma},\vec{\beta}\rangle$, We could simulate the QAOA. So we try to find the best array of $|\vec{\gamma},\vec{\beta}\rangle$ at fixed p. That's exactly what we do in `optimize.ipynb`,finding best array of $|\vec{\gamma},\vec{\beta}\rangle$ at fixed p and n with various methods.

We use dfs to achieve the grid search,hyperopt package to finish Bayesian Optimization and scipy.optimize to implement basinhopping algorithm in L-BFGS-B method. We compare these three algorithm using its F_p.

$$F_p(\vec{\gamma},\vec{\beta}) = \langle\vec{\gamma},\vec{\beta}|C|\vec{\gamma},\vec{\beta}\rangle$$

It turns out that basinhopping and L-BFGS-B have done a faster and better job.

In `optimize.py`, We want to find out how F_p changes as p increases in a fixed graph. So we iterate p in range(1,11) and utilize basinhopping algorithm in L-BFGS-B method to optimize the F_p for each p. It print the result of each p and save the $|\vec{\gamma},\vec{\beta}\rangle$ in `result7_p/p=pi.npy`.

We run `optimize.py` for `grapy8_in.npy`,whose picture is `graphic_n=8.png`. Its result is ploted in `result_8.png` with the help of `draw_result.py`.

We run `optimize.py` for `grapy_in.npy`,whose picture is `graphic_n=7.png`. Its result is ploted in `result_7.png` with the help of `draw_result.py`. We could only simulate quantum circuit with noise in n=7 utlizing qiskit, since the free IBM quantum machine has a maxium qubit of 7 and we use its noise data.

The experiment result shows that F_p increases as p increases and

$$\lim_{p\to\infty} \max_{\vec{\gamma},\vec{\beta}} F_p(\vec{\gamma},\vec{\beta}) = C_{max}(z)$$

Quantum Circuits Simulation

We prepare our qubit with an Hadamard Gate to obtain the mixed state. $U(C,\gamma) = e^{-i\gamma C}$ could be achieved by applying a $Rz(-\gamma)$ and two cx gate at both ends of it. $U(B,\beta) = e^{-i\beta B}$ could be simply implemented deploying $R_x(2\beta)$ gate. In this way, we succeed in constructing the quantum circuit. `circuit.png` is a quantum circuit example at n=7,p=2.

To run the `sim.py`,you need to replace `My_token` with your IBM Quantum account token and change `provider.get_backend('ibm_nairobi')` to the name of an 7 qubits quantum machine in your account.

We obtain our result both with and without noise with the help of 'qasm_simulator'. We draw the top 5 states of each situation in the same histogram for p in range(1,7) . The result is in `figure/`. We try the original 1024 shots and the large enough 100000 shots, which we aim to observe the real quantum computation result and the expectation of the final states individually. At the same time,we plot the Accuracy-p figure with and without noise, to help us better understand the influence of noise.

$$Accuracy = \frac{number\ of\ desired\ states}{total\ shot\ number}$$

Besides, in the experiment we attempt to construct the R_z error ourself, which turn out to be another precision disasters. As show in the `failure.jpg`, we set the possibility of error to zero, but we still obtain an undesired gate result.(It's P(1) is obviously not zero and its P(3) is not 1.) We failed to fix this precision bug. That's why we turn to actual quantum machine noise data.

As we analyze the histogram pictures in figure folder, we will find that for shots=100000, the accuracy states could always be distinguished,though hard, while for shots=1024,they could be figured out at p=2 and 3. It's due to the random nature of quantum measurement. As long as the expectation two states are close, they will mix up easily.

As we analyze Accuracy-p figure, it's easy to obtain that noise dramatically decrease the accuracy. As p increases, the depth of quantum circuit increases,leading to more noise and less accuracy. Besides, although Accuracy increase with p for noiseless situation just as it's theoretically proved , the noise effect will over weigh the accuracy improvement of p and become dominate for sufficient large p (in this case p=2). That's why Accuracy first increase then decrease as p increases.

Summary

1. We implement QAOA on Max-Cut using unitary matrix operator.
2. We tried three different ways of parameters optimization and test them with various n and p to find out the pros and cons.
3. We construct the quantum circuit of QAOA.

4. We simulate the quantum circuit using qiskit and use real quantum machine's error data to simulate the noise.
5. We plot our experiment result and discuss the reason behind it.