Portfolio Optimization Based on QAOA

QHack Open Hackathon Project

cyx

1. Object of Project

We aim to solve the portfolio optimization problem which is one of the most popular topics in the area of finance. The problem can be formulated as a combinatorial optimization problem subject to certain constraints provided below.

$$\min_{x \in \{0,1\}^n} q x^T \Sigma x - \mu^T x$$
$$\text{subject to: } 1^T x = B$$

where we use the following notation:

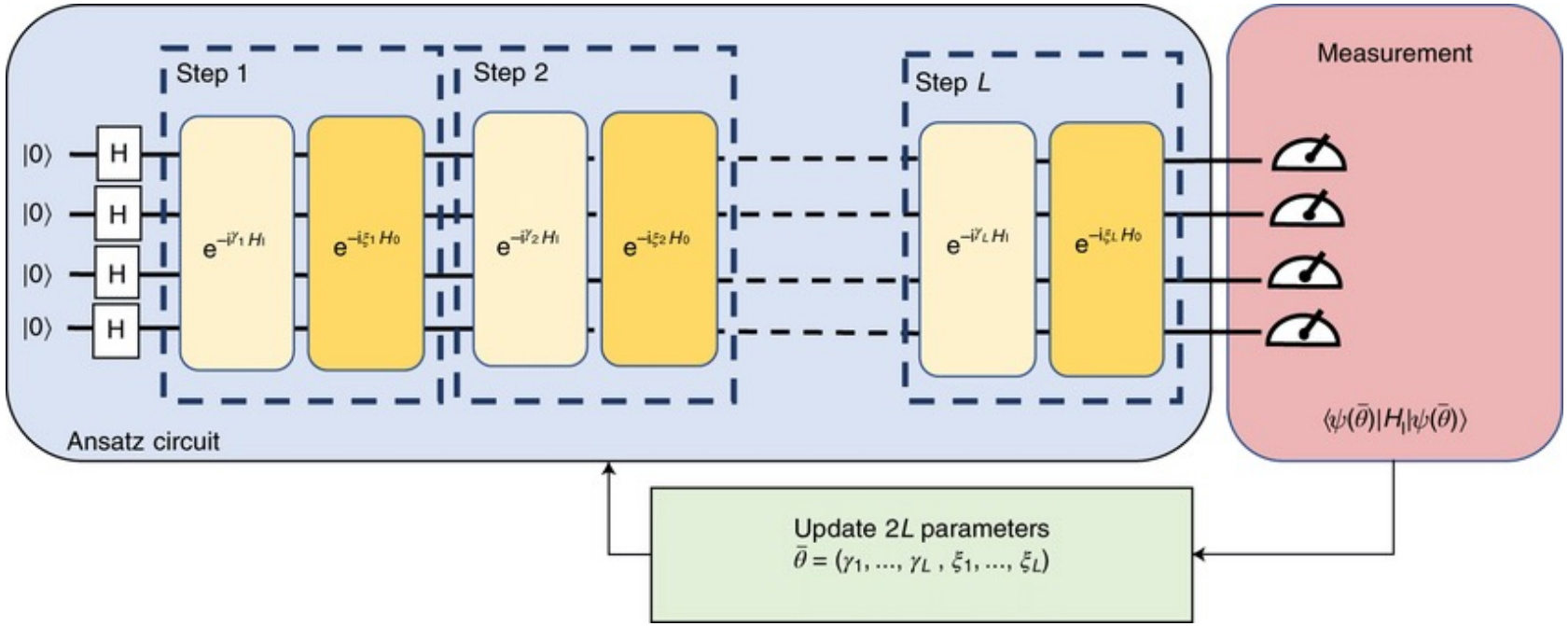
- $x \in \{0,1\}^n$ denotes the vector of binary decision variables, which indicate which assets to pick ($x[i] = 1$) and which not to pick ($x[i] = 0$),
- $\mu \in \mathbb{R}^n$ defines the expected returns for the assets,
- $\Sigma \in \mathbb{R}^{n \times n}$ specifies the covariances between the assets,
- $q > 0$ controls the risk appetite of the decision maker,
- and B denotes the budget, i.e. the number of assets to be selected out of n.

By treating the constraint as a penalty term in the cost function, this problem can be reformulated as a quadratic unconstrained binary optimization (QUBO) problem.

2. Method

- We have exact optimization methods (e.g. Minimum Eigen solver) for solving this type of QUBO problems. But the running times of these method usually grow exponentially with the problem size and thus it would be very expensive and impractical to implement them when dealing with real-world portfolio optimization problems involving a large number of assets.

- In this project, we employ the well-known quantum approximate optimization algorithm (QAOA) to solve portfolio optimization problems and attempt to find the quantum advantage over classical methods. The structure of QAOA is shown below.



3. First-phase Experiments

We perform three experiments for this project. In the first two experiments, we implement the QAOA algorithm with Qiskit and PennyLane respectively. In the last experiment, we select the NumPyMinimumEigensolver as the benchmark method to compare with the QAOA. To make a fair comparison, we use the same server environment and data for all experiments. We provide detailed information of our experiments below.

Experiment	Framework	Optimizer	Backend	Circuit Depth	Max_Iteration	Num_assets	Budget	Penalty
A	Qiskit	QAOA with COBYLA	qasm_simulator (CPU) aer_simulator_statevector (GPU)	3	1000	2-28	num_assets /2	num_assets
B	PennyLane	QAOA with ADAM	default.qubit	3	1000	2-17	num_assets /2	num_assets
C	Numpy	NumPyMinimumEigensolver	None	None	None	2-28	num_assets /2	num_assets

Note:

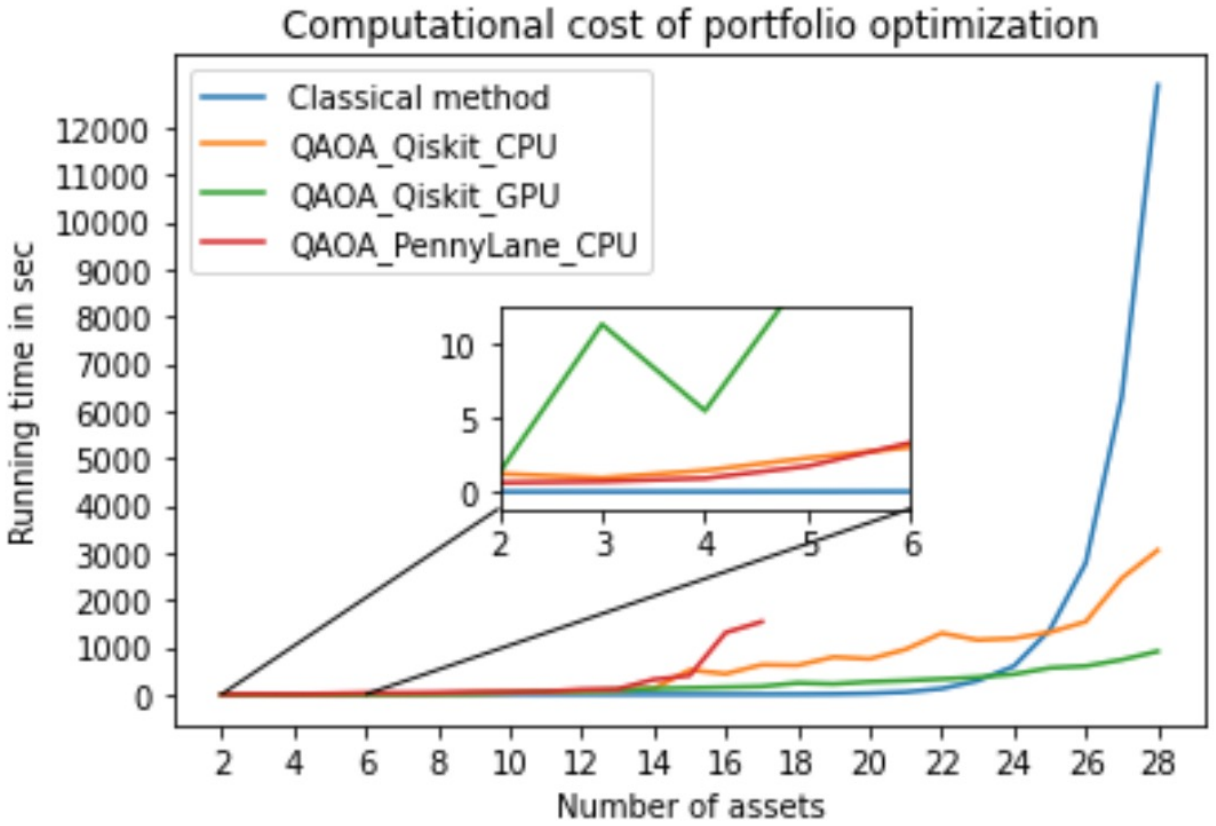
- In Experiment A, we attempted to use Adam optimizer to optimize parameters of QAOA. But the program got stuck when the number of assets is 22. So we used Cobyla optimizer instead for this experiment.

- In Experiment A, we initially used 'aer_simulator_statevector' for CPU simulation. But using it become very expensive when the number of assets is bigger than 16. So we replaced it with 'qasm_simulator' which supports Aer's built-in fast Pauli Expectation.

- In Experiment B, we used backends 'lightening.qubit', 'qiskit.aer' (CPU and GPU), and 'qulacs.simulator' (CPU and GPU) to implement QAOA. But all of them led to much longer running times even for small problem size (e.g. num_assets=2), compared to the 'default.qubit' simulator. So we finally selected 'default.qubit' as the simulator for this experiment.

- In Experiment B, even with the 'default.qubit' simulator, we got extremely long running time when the number of assets become bigger than 17. Thus, we only ran the QAOA program for problems with num_assets ranging from 2 to 17.

4. First-phase Outcomes



Method	Backend	Accuracy*
Classical method	None	100%
QAOA_Qiskit_CPU	qasm_simulator	62.5%
QAOA_Qiskit_GPU	aer_simulator_statevector	62.5%
QAOA_PennyLane_CPU	default.qubit	93.75%

* The accuracy in the table is based on problems with number of assets ranging from 2 to 17

5. Second-phase Experiments

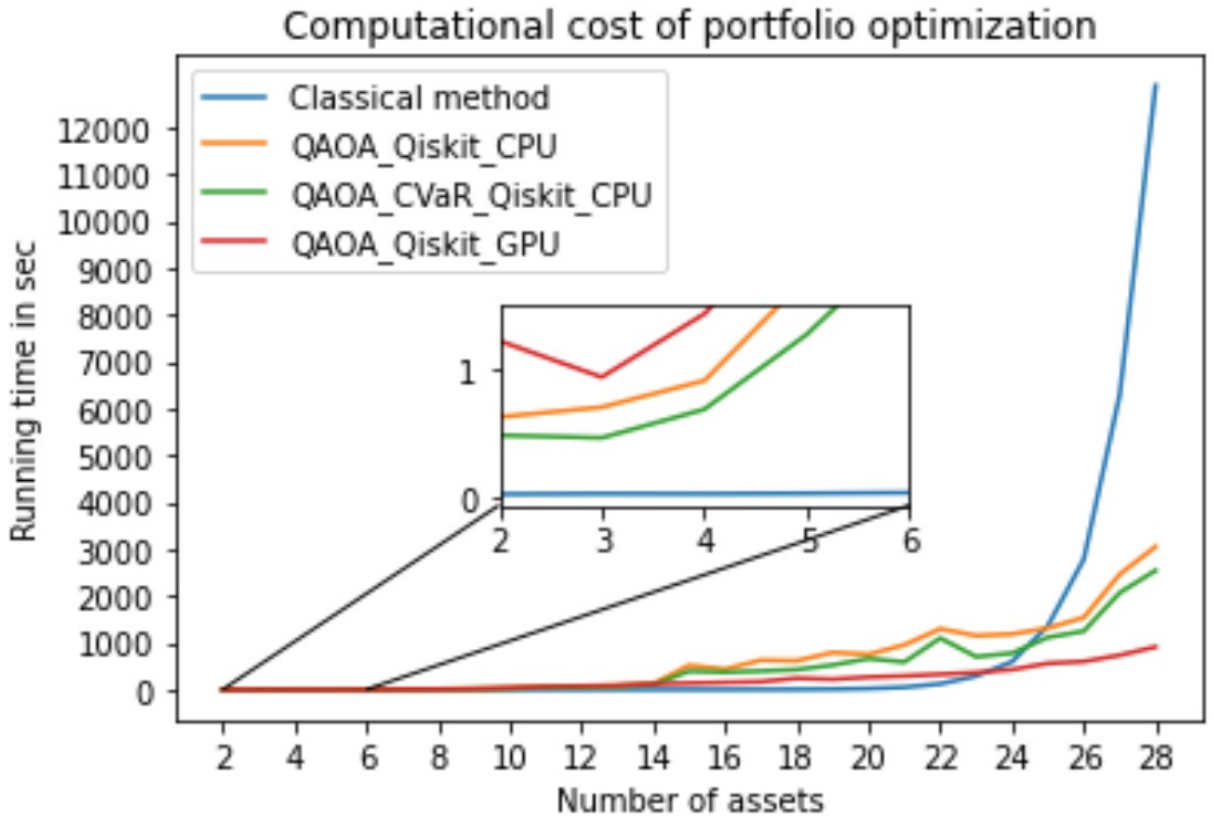
- To tackle the computational issue of PennyLane version of QAOA, we implement it with AWS Bracket SV1 simulator that supports parallel circuit execution. Unfortunately, the program still runs very slowly and the server crashes when the number of assets is bigger than 17.
- To improve the accuracy of QAOA implemented with Qiskit, we employ a variant of QAOA based on Conditional Value-at-Risk (CVaR) proposed in [1]. The main idea is that it is more natural and efficient to optimize the best observed sample of the problem Hamiltonian rather than the expectation value, for problems with diagonal Hamiltonian such as portfolio optimization. So we can replace the original cost function, which is the expectation value of the problem Hamiltonian, with the minimum observed outcome over a set of measurements. However, this will result in a non-smooth cost function which is difficult to optimize. To tackle this problem, the literature [1] proposes the CVaR, a widely used measure in finance, as the cost function.

$$\frac{1}{K}\sum_{k=1}^{K} H_k(\theta) \quad \longrightarrow \quad \frac{1}{\lceil \alpha K \rceil}\sum_{k=0}^{\lceil \alpha K \rceil} H_k.$$

where K is the number of samples and $\alpha \in (0,1]$ is the confidence level. This type of QAOA is expected to converge faster than the original one and enjoy a better performance in obtaining the optimal solution. In this experiment, we run the QAOA_CVaR for $\alpha \in \{0.01,0.1,0.25,0.5,0.75,1\}$ and other hyperparameters remain unchanged.

[1]: Barkoutsos, P. K., et al. “Improving Variational Quantum Optimization using CVaR” .

6. Second-phase Outcomes



Method	Backend	Accuracy*
Classical method	None	100%
QAOA_Qiskit_CPU	qasm_simulator	62.5%
QAOA_Qiskit_GPU	aer_simulator_statevector	62.5%
QAOA_CVaR_Qiskit_CPU*	qasm_simulator	75%
QAOA_PennyLane_CPU	default.qubit	93.75%

* The accuracy in the table is based on problems with number of assets ranging from 2 to 17
* The confidence level α is selected as 0.01 which results in the best model performance

7. Conclusions and Future Work

- In this project, we demonstrate that the QAOA based on Qiskit can achieve the quantum advantage over the classical method in terms of the time complexity for the task of portfolio optimization, while generally suffering from low accuracy. Nevertheless, the problem of low accuracy can be alleviated by computing the CVaR expectation value for QAOA. The CVaR approach has also been shown to speed up the training process of QAOA. In the future, we could finetune more hyperparameters such as the circuit depth to further improve the model accuracy.

- We show that the QAOA based on PennyLane can achieve a very high accuracy for the portfolio optimization problem, compared to Qiskit. Unfortunately, this implementation has a long-running-time issue, especially when the number of assets is bigger than 17. Although we conduct more experiments by leveraging the powerful AWS Bracket SV1 device that supports parallel circuit execution, the computational problem is not resolved. This problem need to be further investigated. For example, we could explore the AWS Bracket Hybrid Jobs feature. We could also attempt to employ the entanglement forging method [2] based on Schmidt decomposition. This method can reduce the required number of qubits by half and thus speed up the algorithm.

[2]: Eddins, Andrew, et al. "Doubling the size of quantum simulators by entanglement forging" .