Proof-of-concept Quantum Use Case

Optimization – Summary

In the current **NISQ (noisy intermediate-scale quantum) era**, the most feasible and promising use cases for quantum computing are **optimization**, **simulation**, and **machine learning**. This is a summarized version of the proof-of-concept use case in optimization produced by DeiC's Quantum Department as part of the Q-Access initiative to make quantum use-case development more accessible to Danish academia and industry. Find the full proof-of-concept use case and more at https://www.deic.dk/q-access.

Optimization
From minimizing risk to maximizing throughput, optimization problems are ubiquitous across many industries, especially in *finance*, *logistics*, and *engineering*.

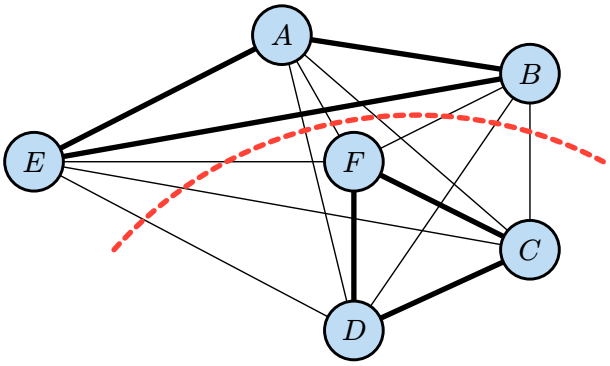
To tackle problems in these areas, most NISQ algorithms take a hybrid approach by using *classical optimization techniques* to tune the parameters of *variational quantum circuits*. In this proof-of-concept use case, we will consider an **optimization problem in logistics** and then walk through, step by step, a variational quantum algorithm which solves the problem.

Problem: Optimizing Last-Mile Routes for a Delivery Company

Suppose that there are two delivery drivers who need to cover six delivery zones labeled A, B, C, D, E, and F. For each pair of zones, there is some cost to both of these zones being placed on the same route, e.g. long distance between the zones, high chance of zig-zagging if one driver serves both zones, delivery time-window conflicts between the zones, etc.

Goal: Assign each zone to one of two routes in such a way that separates pairs that "don't play nicely."

This problem can be formulated as a **max-cut problem** for the graph whose vertices is the set of zones $\mathcal{Z} = \{A, B, C, D, E, F\}$ and whose edges are weighted by the costs. To separate the pairs that don't play nicely we want to consider a *cut*, which is a subset of the edges that once removed separates the vertices into two groups. Moreover, we want to find a cut that maximizes the sum of the cost of the edges in the cut. Below is an example cut that might seem reasonable, separating the zones into a route consisting of $\{A, B, E\}$ and a route consisting of $\{C, D, F\}$.



(1)

Max-cut problems like this can be solved using a variational quantum algorithm called **QAOA (quantum approximate optimization algorithm)** introduced in [1] and [2]. In fact, QAOA is quite general and can be used to solve a wide variety of combinatorial optimization problems, ranging from portfolio optimization in finance to community detection in social networks.

Formulation of Our Problem in a Quantum System

For each of the delivery zones A, B, C, D, E, and F, we first need to know the cost of placing each pair of zones in the same route. We will denote the cost of placing zones A and B in the same route by c_{AB} and so on. The cost of place A and B being on the same route does not depend on the order, so we have $c_{AB} = c_{BA}$. Then, we can define a **cut** in terms of the vertices that it cuts out, which is equivalent to a choosing 0

or 1 for each vertex, where 0 means it is out of the cut and 1 means it is in the cut. So, a cut can be fully described by 6 numbers, which we will denote by

$$\boldsymbol{x} = (x_A, x_B, x_C, x_D, x_E, x_F). \tag{2}$$

Using this, we will define the total cost of the cut to be

$$C(\boldsymbol{x}) = -\frac{1}{2} \sum_{I,J \in \mathcal{Z}} c_{IJ} \cdot (2x_I x_J - x_I - x_J). \tag{3}$$

Our goal then becomes to find the cut $\boldsymbol{x}$ with the *minimum* cost, i.e.

$$\operatorname*{argmin}_{\boldsymbol{x}} C(\boldsymbol{x}). \tag{4}$$

Now, embedding cuts as the states of a quantum system is rather straightforward, since we defined the cuts in terms of terms of 0's and 1's. Each zone will have an associated qubit, and the state corresponding to a cut will be given by

$$|\boldsymbol{x}\rangle := |x_A\rangle \otimes |x_B\rangle \otimes |x_C\rangle \otimes |x_D\rangle \otimes |x_E\rangle \otimes |x_F\rangle. \tag{5}$$

With this formulation, our cost function from Equation (3), becomes the "cost Hamiltonian.""

$$H_C = -\frac{1}{2} \sum_{I,J \in \mathcal{Z}} c_{IJ} \cdot (\mathrm{id}_2 - Z_I \otimes Z_J). \tag{6}$$

A **Hamiltonian** H in quantum mechanics is an operator that represents the total energy of a system, and so our problem becomes finding the lowest energy state, or ground state, of H_C.

Quantum Approximate Optimization Algorithm (QAOA)

The goal of QAOA is to find an approximation of the ground state of a given Hamiltonian, which will give us an approximate solution to our problem. Inspired by quantum annealing, the approach of QAOA is to start with an easy to prepare ground state of the "mixing Hamiltonian" and then evolve the system to the desired ground state.

The key idea is to approximate "adiabatic evolution" of quantum annealing in which a quantum system stays the ground state when evolved slowly enough. To achieve this, QAOA uses a parametrized version of the "first-order Trotterization" given as follows.

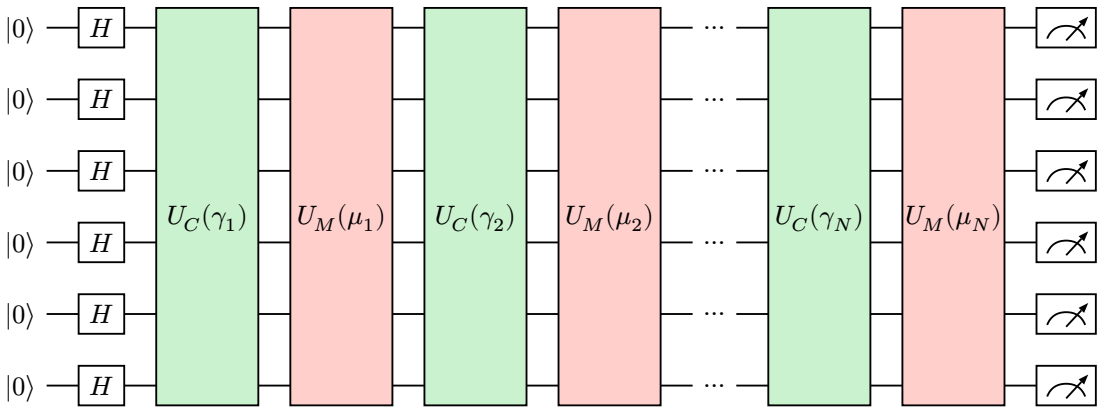
$$U_{\mathrm{QAOA}} = \prod_{k=1}^{N} U_C(\gamma_k) U_M(\mu_k). \tag{7}$$

Here, γ_k and μ_k are parameters which will be optimized. The subscript "M" stands for "mixing" because the operator U_M mixes probability amplitude between the computational-basis states, allowing QAOA to explore the landscape of all possible solutions.

On the other hand the cost operator U_C, which is the unitary evolution determined by the cost Hamiltonian of Equation , is responsible for encoding the cost function in the phases of the qubits, which are then mixed by the mixing operator.

By alternatingly applying the cost and mixing operators, QAOA concentrates the probability distribution near "good" solutions. This process produces a tunable approximation of quantum annealing in the more widely available gate-based model of quantum computation.

The following circuit implements QAOA with a layer of Hadamards to prepare the ground state of the mixing Hamiltonian and alternating subcircuits implementing the cost and mixing operators.

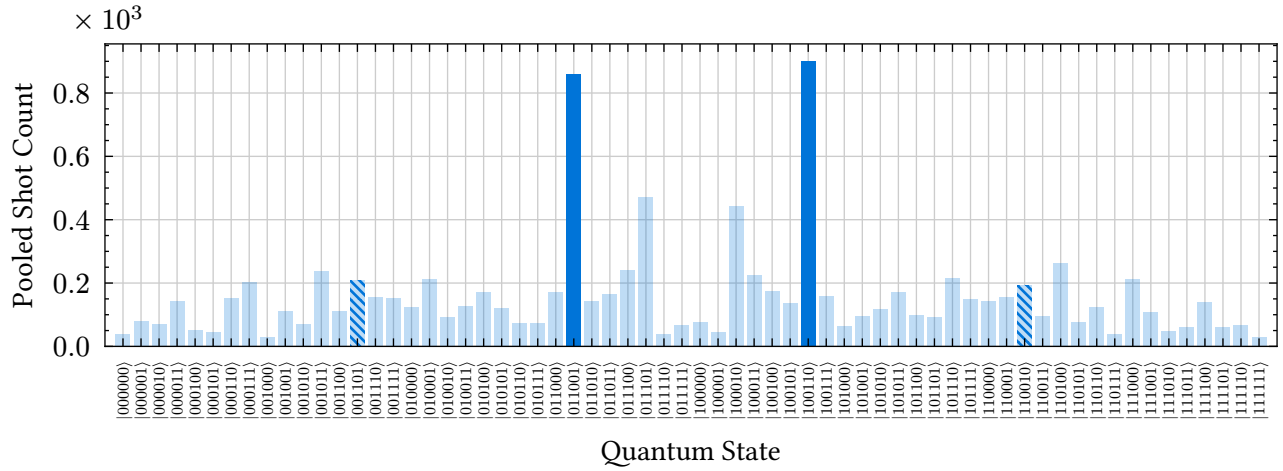


(8)

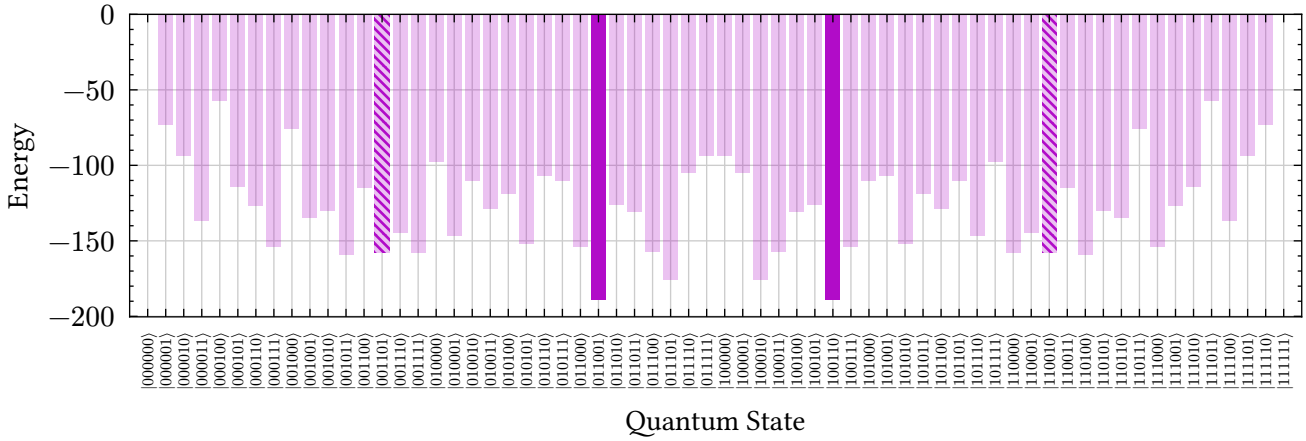
We have implemented the QAOA algorithm tailored to this problem using Qiskit [3] and SciPy [4], which is available at https://gist.github.com/gkpotter/2fb05ac6e901f413295ea67af1ebb7cd. For more information about this implementation, see the full proof-of-concept use case guide at https://www.deic.dk/q-access.

Results

Below are the pooled shot counts for sampling 5 separate optimizations of the above circuit.



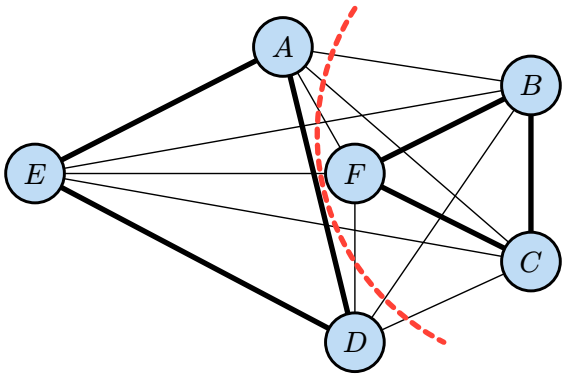
The dashed columns are the counts for our initial guess from Figure (1), which is significantly less frequent than two most frequent outcomes. The shot counts are symmetric because swapping 0's and 1's does not change the corresponding cut of the graph. The solid columns correspond to the two most frequent outcomes, and they do indeed turn out to be the optimal solution. We can directly calculate the energy of each state according to H_C to verify this.



From this plot, we can see that our initial guess is not actually that far off, but there are better cuts including the optimal cut. The two states corresponding to this optimal cut are precisely the ground states of H_C.

$$\mathcal{G}(H_C) = \left\{ \left| \text{  } \right\rangle = |011001\rangle, \left| \text{  } \right\rangle = |100110\rangle \right\} \tag{9}$$

Finally, we can visualize the optimal cut as follows.



(10)

Our initial guess had an energy of -158, whereas the optimal cut has an energy of -189 which is roughly a 20% improvement!

References

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv*, Nov. 2014, doi: 10.48550/arXiv.1411.4028.

[2] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem," *arXiv*, Dec. 2014, doi: 10.48550/arXiv.1412.6062.

[3] A. Javadi-Abhari *et al.*, "Quantum computing with Qiskit," *arXiv*, May 2024, doi: 10.48550/arXiv.2405.08810.

[4] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Feb. 2020, doi: 10.1038/s41592-019-0686-2.

Appendix: Data and Resources

The Python code implementing the QAOA algorithm tailored to this problem using Qiskit [3] and SciPy [4] is available at https://gist.github.com/gkpotter/2fb05ac6e901f413295ea67af1ebb7cd.