Proof-of-concept Quantum Use Case

Optimization

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.

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

Simulation
Simulating quantum systems is a notoriously difficult but powerful tool for solving problems in *chemistry*, *physics*, *material science*, and the *pharmaceutical industry*.

Machine Learning
From *anomaly detection* and *classification problems* to *generating text and images*, machine learning has found countless applications touching almost every sector of society.

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.

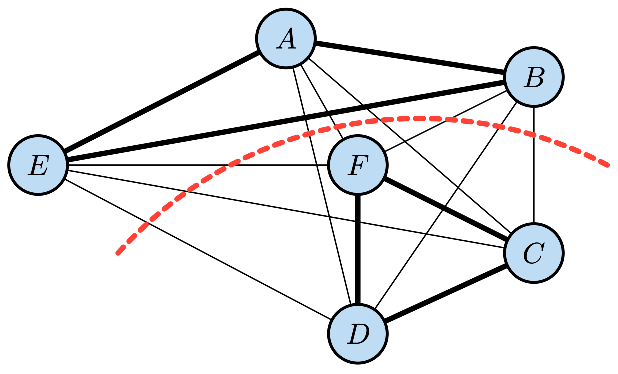
This proof-of-concept use case is part of a series produced by DeiC's Quantum Department as part of the Q-Access initiative to make quantum use-case development more accessible to Danish academia and industry. You can find the proof-of-concept use cases for **simulation** and **machine learning** at https://www.deic.dk/q-access. We will assume a basic familiarity with the quantum circuit model of quantum computing, but we will also try to give as many details as possible.

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 are labeled by 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.

Precise Formulation of Our Problem

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 $c_{AB} = c_{BA}$. Now, suppose that we are given the following costs.

$$\begin{array}{ccccc} c_{AB} = 17 & c_{AC} = 30 & c_{AD} = 10 & c_{AE} = 6 & c_{AF} = 31 \\ & c_{BC} = 10 & c_{BD} = 18 & c_{BE} = 41 & c_{BF} = 12 \\ & & c_{CD} = 9 & c_{CE} = 20 & c_{CF} = 7 \\ & & & c_{DE} = 12 & c_{DF} = 8 \\ & & & & c_{EF} = 15 \end{array} \tag{2}$$

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

$$\boldsymbol{x} = (x_A, x_B, x_C, x_D, x_E, x_F). \tag{3}$$

For example, the cut depicted above would be given by

$$\boldsymbol{x} = (0, 0, 1, 1, 0, 1) \tag{4}$$

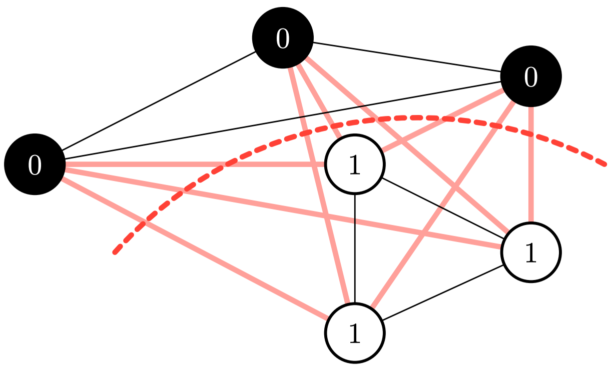
(or equivalently $\boldsymbol{x} = (1, 1, 0, 0, 1, 0)$). Now, if $x_A = 1$ and $x_B = 0$ or vice versa, then A and B are in different routes, and we should include c_{AB} in the cost of the cut. Therefore, we need an expression that is 1 when $x_A \neq x_B$ and 0 when $x_A = x_B$, which is given below.

$$2x_A x_B - x_A - x_B \tag{5}$$

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

$$C(\boldsymbol{x}) = -\frac{1}{2} \sum_{I,J \in Z} c_{IJ} \cdot (2x_I x_J - x_I - x_J). \tag{6}$$

We will see soon the reason for choosing a negative sign, but the $\frac{1}{2}$ is to undo double counting. Now, looking back at Figure (1) above, if we fill in 0 and 1 corresponding to the the cut, we can also see exactly which edges contribute to the cost.



(7)

For this particular cut, we find that

$$C(\boldsymbol{x}) = -(c_{AC} + c_{AD} + c_{AF} + c_{BC} + c_{BD} + c_{BF} + c_{EC} + c_{ED} + c_{EF}) = -158 \tag{8}$$

but there may be better cuts. Our goal then becomes to find the cut $\boldsymbol{x}$ with the *minimum* cost, i.e.

$$\operatorname*{argmin}_{\boldsymbol{x}} C(\boldsymbol{x}). \tag{9}$$

Expressing our optimization problem in terms of finding a minimum will allow us to put it into the framework of **quantum annealing**, but first we need to embed our problem into a quantum system.

Embedding Our Problem in a Quantum System

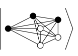
Embedding cuts of a graph as the states of a quantum system is rather straightforward, since we defined the cuts in terms of terms of 0's and 1's. Each zone will have an associated qubit, and the state corresponding to a cut will be given by

$$|\boldsymbol{x}\rangle := |x_A\rangle \otimes |x_B\rangle \otimes |x_C\rangle \otimes |x_D\rangle \otimes |x_E\rangle \otimes |x_F\rangle. \tag{10}$$

For the cut from Figures (1) and (7), the corresponding quantum state is

$$|\boldsymbol{x}\rangle := |0\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \tag{11}$$

which can write more compactly as $|\boldsymbol{x}\rangle = |001101\rangle$. Moreover, for clarity we will occasionally represent a state corresponding to a cut by using a graph similar to the one from Figure (7), as shown below.

$|001101\rangle = \Big|$  $\Big\rangle$ (12)

Next, the cost function can be encoded in a **Hamiltonian**, which we will discuss in more detail in the following section. For now, let us just see how it works in our problem. To translate the cost function from Equation (6), we will use the Z-gate, which is a 1-qubit gate defined by its action on the basis states $|0\rangle$ and $|1\rangle$ as follows.

$$\begin{aligned} Z|0\rangle &= |0\rangle \\ Z|1\rangle &= -|1\rangle \end{aligned} \tag{13}$$

Using two Z-gates. we can reconstruct the cost function as

$$\frac{1}{2}(\mathrm{id}_2 - Z_I \otimes Z_J)|x_I\rangle \otimes |x_J\rangle \tag{14}$$

where I is the identity, since

$$\begin{aligned} \frac{1}{2}(\mathrm{id}_2 - Z_I \otimes Z_J)|0\rangle \otimes |0\rangle &= 0 \\ \frac{1}{2}(\mathrm{id}_2 - Z_I \otimes Z_J)|1\rangle \otimes |0\rangle &= |1\rangle \otimes |0\rangle \\ \frac{1}{2}(\mathrm{id}_2 - Z_I \otimes Z_J)|0\rangle \otimes |1\rangle &= |0\rangle \otimes |1\rangle \\ \frac{1}{2}(\mathrm{id}_2 - Z_I \otimes Z_J)|1\rangle \otimes |1\rangle &= 0 \end{aligned} \tag{15}$$

(which can be seen by changing variables to $z_I = 2x_I - 1$, mapping $0 \mapsto -1$ and $1 \mapsto 1$). Thus, our cost function from Equation (6), becomes

$$H_C = -\frac{1}{2} \sum_{I,J \in \mathcal{Z}} c_{IJ} \cdot (\mathrm{id}_2 - Z_I \otimes Z_J). \tag{16}$$

For the cut $\boldsymbol{x}$ from Figures (1) and (7), whose state is given by Equation (11), we find that

$$H_C|\boldsymbol{x}\rangle = H_C|001101\rangle = -158 \cdot |001101\rangle \tag{17}$$

which aligns with cost from Equation (8)!

Warning: Equation (16) is not a *normalized* quantum state, so it is not a problem that $|-158| > 1$ as it does not represent a probability. In fact, after being normalized, the state would just be $|001101\rangle$ with probability 1. Moreover, H_C is not a *unitary operator* but a *Hermitian operator*. For now, what is important is that it properly encodes the cost function, and we will see in the subsequent sections how H_C is related to a unitary operator that will help us solve our problem.

Motivation from Quantum Annealing

The aim of this section is to provide a motivation for QAOA, but it this is impossible to do without introducing a certain level of formalism, so we have tried to make this section self-contained.

A **Hamiltonian** H in quantum mechanics is an operator that represents the total energy of a system. From a "spectral decomposition" of the Hamiltonian, one can find the possible energy levels of the system, which will be a finite set $\{E_0, E_1, E_2, ..., E_n\}$ typically ordered from smallest to largest. The **ground state** $|\Psi_0\rangle$ of a quantum system is the lowest energy state, which will have energy E_0 and satisfy

$$H|\Psi_0\rangle = E_0|\Psi_0\rangle. \tag{18}$$

The **excited states** $\{|\Psi_1\rangle, |\Psi_2\rangle, ..., |\Psi_n\rangle\}$ are the states with energy greater than the ground state, and they satisfy $H|\Psi_i\rangle = E_i|\Psi_i\rangle$. Looking back at Equation (17), in this new language, we would say that the state $|001101\rangle$ has energy -158, but it is not clear if this is a ground state or an excited state.

In general the Hamiltonian might depend on time, in which case we would write $H(t)$. Then, at each instant in time, there is an **instantaneous ground state** and **instantaneous excited states**. As time progresses, quantum systems will tend to stay in the ground state, if evolved slowly enough and in such a way that does not introduce enough energy into the system to transition it to an excited state. In this case, we say that system is evolving **adiabatically**.

The problem of determining ground states is a very computationally difficult problem in general, and **quantum annealing** is a specialized quantum algorithm that takes advantage of adiabatic evolution to find these ground states starting from an easy-to-construct ground state (under some conditions[1]).

Outline of Quantum Annealing

Goal: Prepare the ground state $\left|\Psi_0^{\text{target}}\right\rangle$ of a given Hamiltonian H_{target}.

Strategy

1. Initialize the system in an easy-to-construct ground state $\left|\Psi_0^{\text{initial}}\right\rangle$ of a Hamiltonian H_{initial}.
2. Evolve the system according to the time-dependent Hamiltonian

$$H(t) = (1-t) \cdot H_{\text{initial}} + t \cdot H_{\text{target}} \tag{19}$$

   from time $t = 0$ to time $t = 1$ slowly enough that the system evolves adiabatically.
3. The system will now be in the ground state $\left|\Psi_0^{\text{target}}\right\rangle$ of $H(1) = H_{\text{target}}$ as desired.

This leaves two big questions unanswered. How do we evolve a system adiabatically according to a given time-dependent Hamiltonian and how can we implement this on a gate-based quantum computer? Let us tackle the first question, because it should shed some light on the second question, and hopefully make the path towards developing QAOA feel more natural.

For a time-independent Hamiltonian, solving Schrödinger's equation gives a unitary operator that evolves the system, which is given by

[1]There must be a gap between the ground state and the lowest energy excited state throughout the evolution of the system. Moreover, "slowly" in this context is defined in terms of this gap and the uncertainty principle to ensure that the gap is never crossed.

$$U(t) = \exp(-itH). \tag{20}$$

If the Hamiltonian is given by one of the Puali operators, i.e the X-gate, Y-gate, and Z-gate, then this unitary evolution operator is a rotation of the Bloch sphere around the x-axis, y-axis, and z-axis, respectively.

$$\begin{aligned} \exp(-itX) &= R_X(2t) \\ \exp(-itY) &= R_Y(2t) \\ \exp(-itZ) &= R_Z(2t) \end{aligned} \tag{21}$$

Moreover, these rotation operators are in the native gate set of a number of quantum computing platforms, so implementing the evolution of Hamiltonians built out of Pauli operators is often quite feasible. Now, returning to our problem, finding the ground state of the cost Hamiltonian H_C from Equation (16) is equivalent to finding the ground state of

$$\tilde{H}_C = \frac{1}{2} \sum_{I,J \in \mathcal{Z}} c_{IJ} \cdot Z_I \otimes Z_J. \tag{22}$$

obtained by dropping the constant terms. Indeed, these constant terms would just introduce a global phase in the unitary evolution operator, which we can ignore. For this modified Hamiltonian $\tilde{H}_C$, the unitary evolution operator is given by

$$\tilde{U}_C(t) = \sum_{I,J \in \mathcal{Z}} R_{Z_I Z_J}(c_{IJ} \cdot t). \tag{23}$$

Note that $R_{ZZ}(t) \neq R_Z(t) \otimes R_Z(t)$, but instead $R_{ZZ}(t) = R_Z(t) \oplus R_Z(-t)$. However, the R_{ZZ} gate is still a native gate for a number of quantum computing platforms and otherwise can be implemented with two $CNOT$ gates and a single R_Z gate.

Unfortunately, the precise description of the unitary evolution operator for a time-dependent Hamiltonian is not so simple, but we can employ a discretization procedure called **Trotterization**. For our problem, the time-dependent Hamiltonian is

$$H(t) = (1-t) \cdot H_{\text{initial}} + t \cdot H_C. \tag{24}$$

Now in order to discretize, we first need to divide our time interval from $t = 0$ to $t = 1$ into N intervals of length $\frac{1}{N}$. The endpoints of these time intervals are given by

$$t_0 = 0, t_1 = \frac{1}{N}, t_2 = \frac{2}{N}, ..., t_N = 1. \tag{25}$$

The difficulty arises because in general H_{initial} will not "commute" with H_{target}, or in this case H_C, which means that the order we apply them matters. Taking into account this time-ordering at each step is necessary to approximate the correct unitary evolution. The ansatz considered in QAOA, is inspired by the "first-order" Trotterization of the unitary evolution operator, which in this case is given by

$$U(1) \approx \prod_{i=k}^{N} \exp(-i(1-t_k)H_{\text{initial}}) \exp(-it_k H_C) = \prod_{i=k}^{N} U_{\text{initial}}(1-t_k) U_C(t_k). \tag{26}$$

In each interval, this approximation applies both Hamiltonians alternatingly to approximate the evolution of the fully time-dependent Hamiltonian of Equation (24). Indeed, as $N \to \infty$, this approximation converges to $U(1)$ with an error scaling of $O\left(\frac{1}{N}\right)$. There are higher-order Trotterizations as well with better error scaling in terms of N, but usually worse depth scaling per time step. Moreover, in QAOA, these time intervals are replaced with parameters, which can be optimized to significantly increase convergence compared to the uniform discretization.

Quantum Approximate Optimization Algorithm (QAOA)

The goal of QAOA is to find an approximation of the ground state of a given Hamiltonian, which as discussed above is exactly what we need to solve our problem. Just as with quantum annealing, the approach of QAOA is to start with an easy to prepare ground state of some Hamiltonian and then evolve the system to the desired ground state. Instead of evolving this directly, QAOA uses a parametrized version of the "first-order Trotterization" from Equation (26).

The typical initial ground state is the uniform superposition of all possible states. This can be prepared using a single layer of Hadamard gates. The Hadamard gate is a 1-qubit gate give by

$$H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{27}$$

which when applied to the state $|0\rangle$ gives a uniform superposition of $|0\rangle$ and $|0\rangle$. When you measure the state $|+\rangle$, 50% of the time you will measure 0 and 50% of the time you will measure 1. Moreover, by applying a Hadamard gates to each of two qubits in the state $|00\rangle$ gives a uniform superposition of the 4 possible standard "computational-basis states" for 2 qubits.

$$H \otimes H|00\rangle = |{+}{+}\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \tag{28}$$

Finally, for our problem, since we have 6 qubits, we will need to apply 6 Hadamard gates, one to each to qubit.

$$|\Psi_0^M\rangle = H \otimes H \otimes H \otimes H \otimes H \otimes H|000000\rangle = |{+}{+}{+}{+}{+}{+}\rangle \tag{29}$$

This gives the uniform superposition of the 64 possible states, which encodes the 64 possible solutions to our problem as shown in Equation (38) below. Now, the Hamiltonian for which this uniform superposition is the ground state is simply given the negative sum of X gates, since $-X|+\rangle = -|+\rangle$.

$$H_M = -\sum_{I \in \mathcal{Z}} X_I = -(X_A + X_B + X_C + X_D + X_E + X_F) \tag{30}$$

Finally, the evolution operator for this Hamiltonian is given as follows.

$$\begin{aligned} U_M(t) &= \bigotimes_{I \in \mathcal{Z}} R_{X_I}(-2t) \\ &= R_{X_A}(-2t) \otimes R_{X_B}(-2t) \otimes R_{X_C}(-2t) \otimes R_{X_D}(-2t) \otimes R_{X_E}(-2t) \otimes R_{X_F}(-2t) \end{aligned} \tag{31}$$

The subscript "M" throughout stands for "mixing" because this operator mixes probability amplitude between the computational-basis states, allowing QAOA to explore the landscape of all possible solutions. On the other hand the unitary evolution operator $U_{C(t)}$ for the cost Hamiltonian of Equation (23) is responsible for encoding the cost function in the phases of the qubits, which are then mixed by the mixing operator. The cost unitary is diagonal in the computational basis, which means it does not change the probabilities directly.

By alternatingly applying the cost and mixing operators, QAOA concentrates the probability distribution near "good" solutions. This process approximates quantum annealing, and the full operator, shown below, can be viewed as a parametrized version of the first-order Trotterization of Equation (26).[2]

$$U_{\mathrm{QAOA}} = \prod_{k=1}^{N} U_C(\gamma_k) U_M(\mu_k). \tag{32}$$

[2]The cost operator is applied first in QAOA, because applying $U_{M(t)}$ to the initial state $|{+}{+}{+}{+}{+}{+}\rangle$ has no observable effect, as it only adds a global phase.

Outline of QAOA

Goal: Find an approximation of the ground state $\left|\Psi_0^C\right\rangle$ of a cost function Hamiltonian H_C as in Equation (16) (or $\tilde{H}_C$ as in Equation (22)).

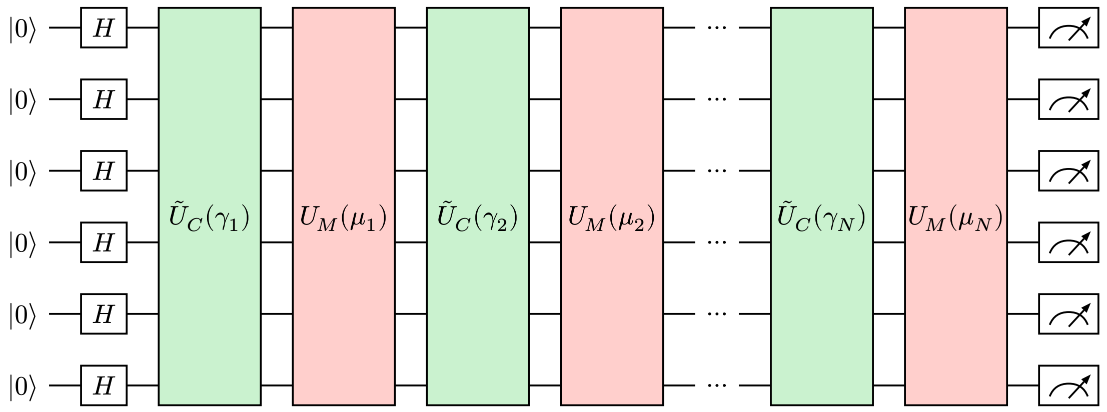
Strategy

1. Construct a variational quantum circuit depending on $2N$ parameters $\gamma_1, \mu_1, \gamma_2, \mu_2, ..., \gamma_N, \mu_N$ as follows.
   i. Initialize the state $|+\rangle$ on each qubit.
   ii. Alternatingly apply circuits for the cost and mixing operators N times with the parameters γ_k and μ_k to apply U_{QAOA} of Equation (32).
   iii. Measure all the qubits.
2. Optimize the parameters, by repeatedly running the circuit from Step 1 multiple times to estimate the cost of the prepared state according to H_C and updating the parameters to lower this cost.
3. Sample the final optimized circuit multiple times and output the most frequently measured states.

Now, we will discuss the construction of the variational quantum circuit of Step 1 in the next section, but this still leaves a big question unanswered concerning Step 2. How do we actually optimize the parameters? There is a plethora of classical optimization algorithms which could be used for this step of QAOA. However, we will not delve into these, but instead rely on existing implementations from the open-source Python library SciPy [3].

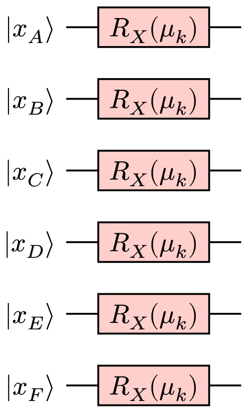
Constructing a Quantum Circuit for QAOA

Following the outline of QAOA above, the variational circuit has the following form.



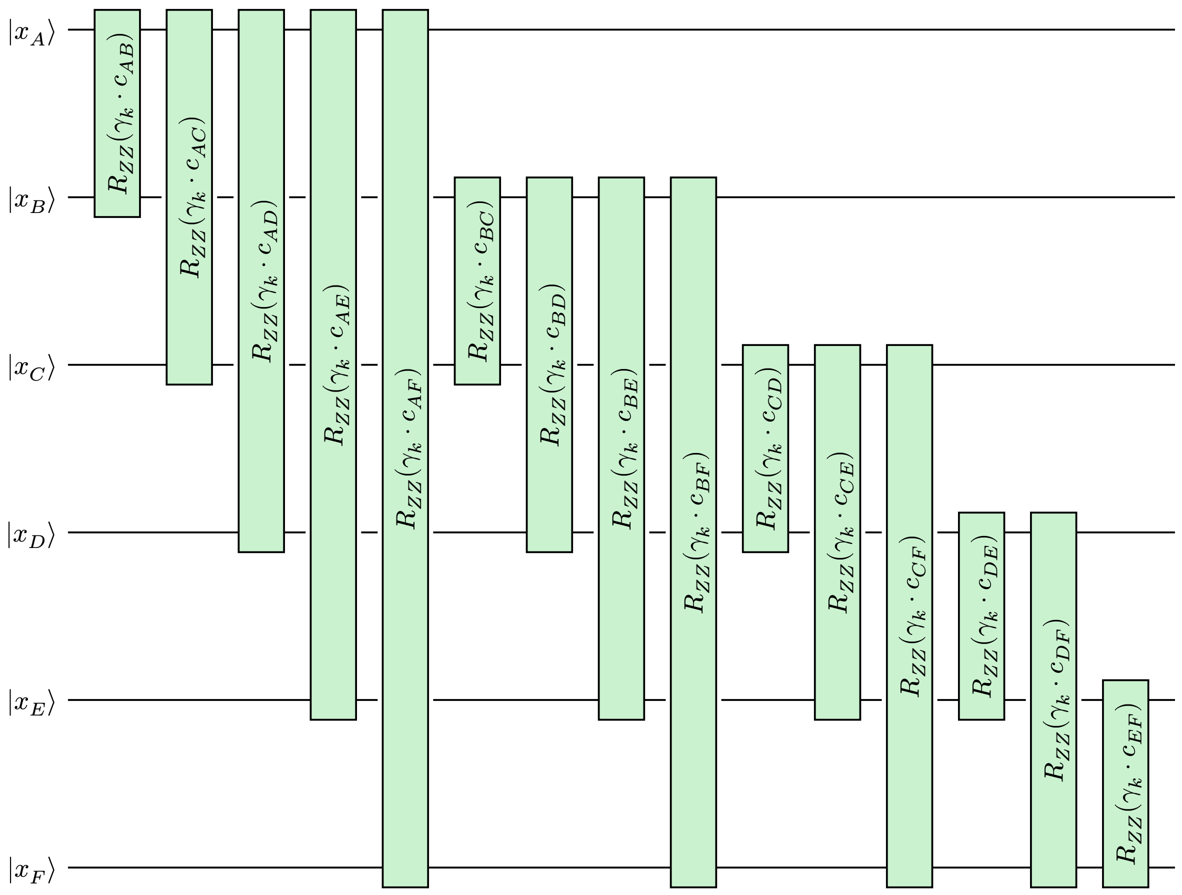
(33)

Here, H stands for the Hadamard gate. Now, the subcircuit for mixing operator is implemented using R_X-gates as follows.



(34)

Finally, the subcircuit for the cost operator is implemented using R_{ZZ}-gates as follows.


$|x_A\rangle$
$|x_B\rangle$
$|x_C\rangle$
$|x_D\rangle$
$|x_E\rangle$
$|x_F\rangle$
$R_{ZZ}(\gamma_k \cdot c_{AB})$
$R_{ZZ}(\gamma_k \cdot c_{AC})$
$R_{ZZ}(\gamma_k \cdot c_{AD})$
$R_{ZZ}(\gamma_k \cdot c_{AE})$
$R_{ZZ}(\gamma_k \cdot c_{AF})$
$R_{ZZ}(\gamma_k \cdot c_{BC})$
$R_{ZZ}(\gamma_k \cdot c_{BD})$
$R_{ZZ}(\gamma_k \cdot c_{BE})$
$R_{ZZ}(\gamma_k \cdot c_{BF})$
$R_{ZZ}(\gamma_k \cdot c_{CD})$
$R_{ZZ}(\gamma_k \cdot c_{CE})$
$R_{ZZ}(\gamma_k \cdot c_{CF})$
$R_{ZZ}(\gamma_k \cdot c_{DE})$
$R_{ZZ}(\gamma_k \cdot c_{DF})$
$R_{ZZ}(\gamma_k \cdot c_{EF})$
(35)

So, the entire QAOA circuit requires just H-gates, R_X-gates, and R_{ZZ}-gates. In the next section, we will see how to implement this circuit in Qiskit along with the QAOA algorithm. When it comes to hardware implementations, not every machine can natively implement these gates, so this circuit will require transpilation. In the appendices, we briefly discuss how to modify this circuit to run on "VLQ," which is a 24-qubit superconducting quantum computer from IQM whose native gate set consists of R_X and CZ.

Qiskit Implementation of QAOA for Our Problem

First, we need to import everything we need from Qiskit [4] and SciPy [3].

```python
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit import Parameter
from qiskit.quantum_info import SparsePauliOp

from qiskit_aer.primitives import Estimator
from qiskit_aer import AerSimulator

from scipy.optimize import minimize
from scipy.constants import pi
```

Then, we input our problem as a cost matrix and specify the number of discretization steps our QAOA circuit should have.

```python
cost_matrix = [
  [ 0, 17, 30, 10,  6, 31],
  [17,  0, 10, 18, 41, 12],
  [30, 10,  0,  9, 20,  7],
  [10, 18,  9,  0, 12,  8],
  [ 6, 41, 20, 12,  0, 15],
  [31, 12,  7,  8, 15,  0],
]

steps = 5
```

Then, we define our mixing and cost operator subcircuits.

```python
def cost_operator_subcircuit(n, cost_matrix, param):
    sub = QuantumCircuit(n, name="U_C")

    for i in range(n):
        for j in range(i + 1, n):
            sub.rzz(cost_matrix[i][j] * param, i, j)

    return sub.to_gate()

def mixing_operator_subcircuit(n, param):
    sub = QuantumCircuit(n, name="U_M")
    sub.rx(param, range(n))

    return sub.to_gate()
```

Next, we set up our parameters and build the variational circuit using the cost and mixing operator subcircuits.

```python
n = len(cost_matrix)
gamma = [Parameter(f"γ{i}") for i in range(steps)]
mu = [Parameter(f"μ{i}") for i in range(steps)]

qr = QuantumRegister(n, name="x")
cr = ClassicalRegister(n)
qc = QuantumCircuit(qr, cr)

qc.h(qr)

for i in range(steps):
    qc.append(cost_operator_subcircuit(n, cost_matrix, gamma[i]), qr)
    qc.append(mixing_operator_subcircuit(n, mu[i]), qr)

qc.measure(qr, cr)
```

Then, we set up our cost Hamiltonian ($\tilde{H}_C$) and define the cost function that we want to minimize. Here, we are using the Qiskit Aer simulator to estimate the energy of the state produced by our circuit.

```python
max_cost = max(map(max, cost_matrix))

H_C = 0.5 * SparsePauliOp.from_list(
    [
        (
            "I" * (i) + "Z" + "I" * (j - i - 1) + "Z" + "I" * (5 - j),
            cost_matrix[i][j] / max_cost,
        )
        for i in range(n)
        for j in range(i + 1, n)
    ]
)
```

```python

estimator = Estimator()

def cost(params):
    val = estimator.run(qc, H_C, parameter_values=params).result().values[0]
    return val
```

Then, we choose some initial parameters and a classical optimization method, and we optimize our circuit.

```python
initial_params = [pi] * steps + [pi / 2] * steps
optimum = minimize(cost, x0=initial_params, method="Powell")

optimal_params = optimum.x
optimal_qc = qc.assign_parameters(optimal_params)
```

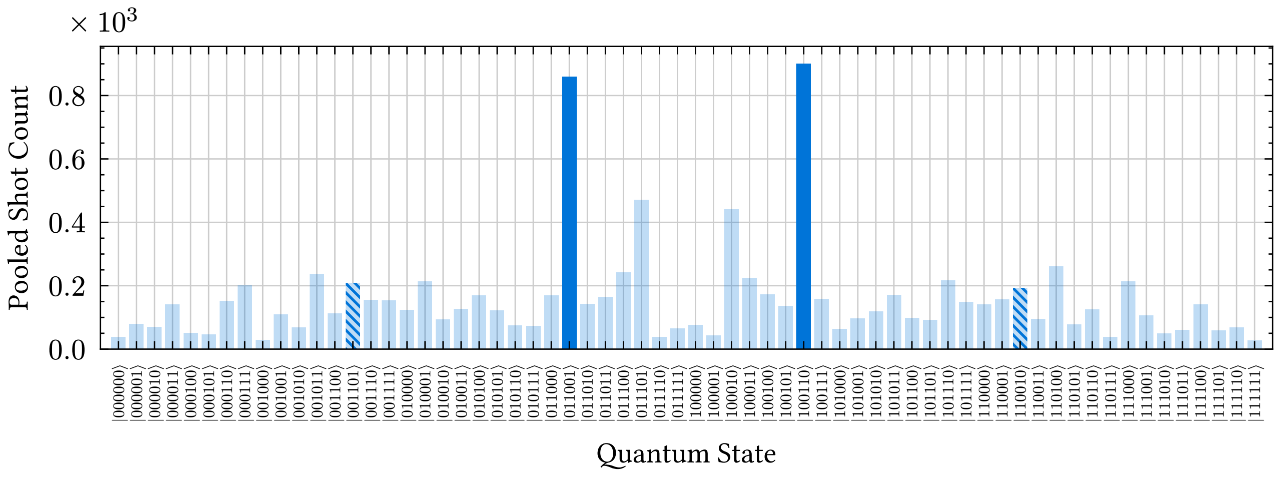
Finally, we sample this optimized circuit and record the measurement counts. Our approximate solutions will be the most frequently measured outcomes. Again, we are using the Qiskit Aer simulator to sample our final circuit.

```python
sim = AerSimulator()
result = sim.run(optimal_qc.decompose(), shots=1_000).result()
counts = result.get_counts()
```

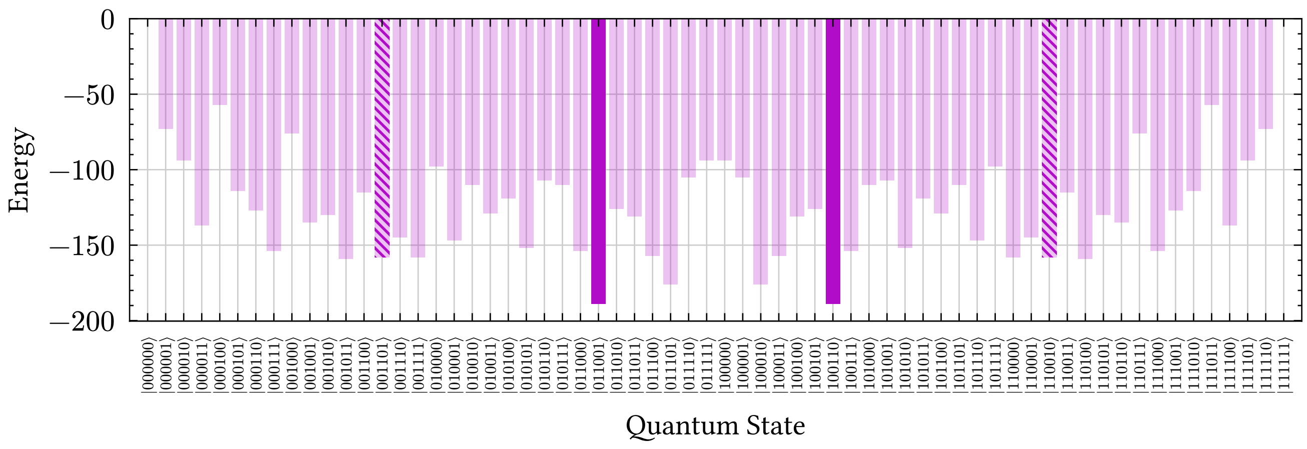
In the next section, we will discuss the results of this simulation, and see how well we are able to solve our problem.

Results

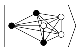
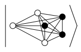
Below are the pooled shot counts for sampling 5 separate optimizations of the circuit implemented in the above Qiskit code. Instability is a perenniel problem in optimization often requiring careful tweaking of meta-parameters for each specific optimization algorithm and problem. In this case, to dampen any outlying runs, we decided to use run pooling rather than get into the weeds of tweaking meta-parameters, but each problem could require different techniques for fine-tuning.



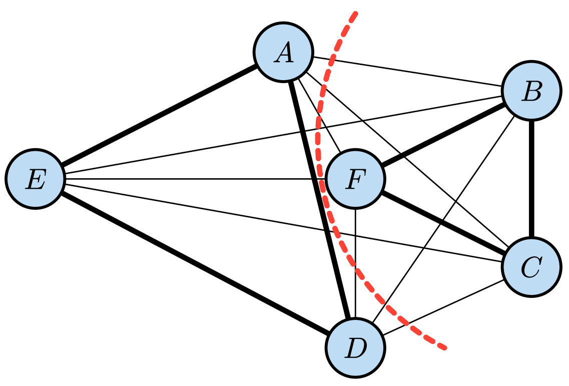
The dashed columns are the counts for our initial guess from Figure (1), which is significantly less frequent than two most frequent outcomes. The shot counts are symmetric because swapping 0’s and 1’s does not change the corresponding cut of the graph. The solid columns correspond to the two most frequent 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 (and therefore als $\tilde{H}_C$)

$\mathcal{G}(H_C) = \Big\{ \Big|$  $\Big\rangle = |011001\rangle, \Big|$  $\Big\rangle = |100110\rangle \Big\}$ (36)

Finally, we can visualize the optimal cut as follows.



(37)

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.

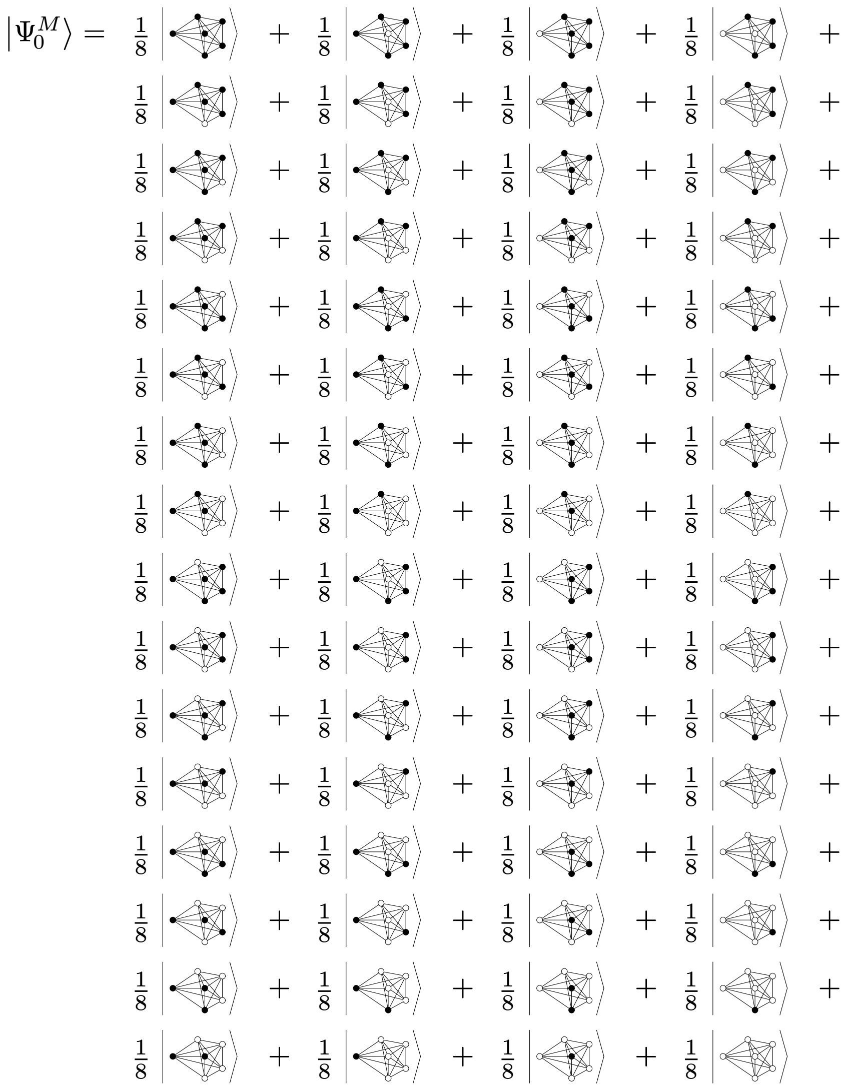
[2] E. Farhi, J. Goldstone, and S. Gutmann, “A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem,” *arXiv*, Dec. 2014, doi: 10.48550/arXiv.1412.6062.

[3] 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.

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

Appendix: Ground State of Mixing Hamiltonian

The ground state $\left|\Psi_0^M\right\rangle$ of the mixing Hamiltonian H_M is the superposition of all possible cuts with equal weighting, as depicted below.



(38)

The $\frac{1}{8}$ weighting comes from the fact that each of the 64 states has a probability of $\frac{1}{64} \approx 1.5\%$ of appearing when measuring this state.

Appendix: Implementation on IQM Star 24

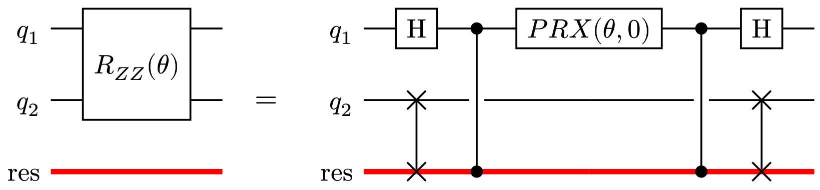
The IQM Star 24 is a 24-qubit QPU with superconducting transmon qubits and a star-shaped topology. The native single-qubit operation is the PRX-gate, which has two parameters.

$$PRX(\theta, \varphi) = R_Z(\varphi) R_X(\theta) R_Z(-\varphi) \tag{39}$$

From this it is possible to define each of the rotation gates R_X, R_Y, and R_Z as follows.

$$\begin{aligned} R_X(\theta) &= PRX(\theta, 0) \\ R_Y(\theta) &= PRX(\theta, \pi/2) \\ R_Z(\theta) &= PRX(-\pi, 0) PRX(\pi, -\theta/2) \end{aligned} \tag{40}$$

Each qubit is coupled with a central resonator, and the native two qubit operation is the controlled-Z-gate (or CZ-gate) between any qubit and the central resonator. So, we if want to implement the above circuit for QAOA, we need still need to implement R_{ZZ}-gates using PRX-gates and CZ-gates, which can be done as folows.



(41)

However, this still leaves a number of Hadamard gates, which can be implemented using two PRX-gates (up to a global phase). Now, the Hadamard gate is self-adjoint which means that applying the Hadamard gate twice does nothing at all. So, almost all of the Hadamard gates can be simplified away. In the end, the number of PRX-gates and CZ-gates is given as follows.

$$\begin{aligned} N_{PRX} &= 2\,|V| + (|V| + |E|) \cdot N \\ N_{CZ} &= 2|E| \cdot N \end{aligned} \tag{42}$$

Here, $|V|$ is the number of vertices in the graph, $|E|$ is the number of edges in the graph, and N is the number of discretization steps in the QAOA algorithm. For our example, we have $|V| = 6$, $|E| = 15$ and $N = 5$. So, we have the following gate counts.

$$\begin{aligned} N_{PRX} &= 117 \\ N_{CZ} &= 150 \end{aligned} \tag{43}$$

Appendix: Data and Resources

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