QAOA implementation for Airlines Problem

February 8, 2026

Till now, we converted the problem into QUBO/ising model. Our goal is now to implement Quantum Approximate Optimization Algorithm (QAOA) to find the minimum energy for the constructed Hamiltonian or to find the optimum route that satisfies all the defined constraints for our exact cover problem.
Now let's take a simple implementable problem to understand how it works.

1 Problem setup

We have 3 cities: City A, City B, City C.
Set of flights, Routes and Tails are defined as:

- Tails: T: $\{T_1, T_2, T_3\}$
- Flights F: $\{F_1, F_2, F_3, F_4\}$.
  where: $(F_1 : A \text{ to } B)$, $(F_2 : A \text{ to } C)$, $(F_3 : B \text{ to } A)$, $(F_4 : C \text{ to } A)$
- Routes R: $\{R_1, R_2, R_3, R_4\}$

$$R_1 = \{F_1, F_3\}, \qquad R_2 = \{F_2, F_4\}, \qquad R_3 = \{F_1, F_4\}, \qquad R_4 = \{F_2, F_3\}.$$

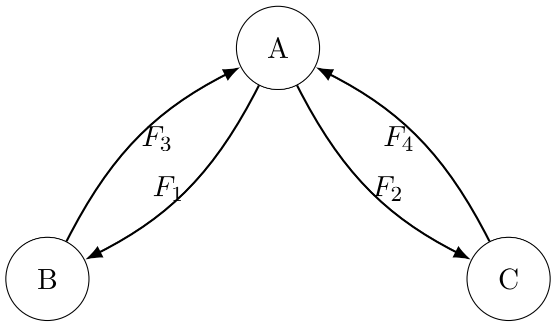

Figure 1: Flight graph for cities A, B, C and flights $F_1, \ldots, F_4$.

The incidence matrix a_{fr} (flight–route membership) is

$$a_{fr} = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ \hline F_1 & 1 & 0 & 1 & 0 \\ F_2 & 0 & 1 & 0 & 1 \\ F_3 & 1 & 0 & 0 & 1 \\ F_4 & 0 & 1 & 1 & 0 \end{array}.$$

The aircraft–route incidence matrix b_{tr} is

$$b_{tr} = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ \hline \text{Tail A} & 1 & 0 & 1 & 0 \\ \text{Tail B} & 0 & 1 & 0 & 1 \end{array}.$$

Decision variables

$$x_r \in \{0,1\} \quad \text{for each } r \in R \qquad (\text{e.g., } x_1, x_2, x_3, x_4).$$

Constraints

The constraints of the problem are defined as:

- For each flight $f \in F$, exactly one selected route contains it:

$$\sum_{r\in R} a_{fr}\, x_r + u_i = 1.$$

- For each aircraft (tail) $t \in T$, it is assigned to exactly one route:

$$\sum_{r\in R} b_{tr} x_{tr} + v_i = 1.$$

Flight coverage (Exact Cover).

$$\begin{aligned} F_1: &\quad x_1 + x_3 + u_1 = 1, \\ F_2: &\quad x_2 + x_4 + u_2 = 1, \\ F_3: &\quad x_1 + x_4 + u_3 = 1, \\ F_4: &\quad x_2 + x_3 + u_4 = 1. \end{aligned}$$

Aircraft (tail) constraints.

$$\begin{aligned} \text{Tail A:} &\quad x_1 + x_3 + v_1 = 1, \\ \text{Tail B:} &\quad x_2 + x_4 + v_2 = 1. \end{aligned}$$

Energy / objective

$$E(x,u,v) = \underbrace{\sum_{r\in R} c_r x_r}_{\text{true route cost}} + \underbrace{\sum_{f\in F} c_f u_f}_{\text{uncovered-flight penalty (weighted)}} + \underbrace{E_{\text{pen}}(x,u,v)}_{\text{hard constraint penalties}} .$$

$$E(x,u,v) = \sum_{r\in R} c_r x_r + \sum_{f\in F} C_f u_f + A \sum_{f\in F} \left[\sum_{r\in R} a_{fr} x_r + u_f - 1 \right]^2 + B \sum_{t\in T} \left[\sum_{r\in R} b_{tr} x_r + v_t - 1 \right]^2 .$$

Using the slack-variable substitution

$$u_f = 1 - \sum_{r\in R} a_{fr} x_r, \qquad u_f^2 = u_f \quad (u_f \in \{0,1\}),$$

we obtain

$$A \sum_{f\in F} \left[\sum_{r\in R} a_{fr} x_r + u_f - 1 \right]^2 = A \sum_{f\in F} \left[-\left(\sum_{r\in R} a_{fr} x_r \right)^2 + 2 \sum_{r\in R} a_{fr} x_r \right]$$
$$= -2A \sum_{f\in F} \sum_{r<r'} a_{fr} a_{fr'}\, x_r x_{r'} + 2 \sum_{r\in R} a_{fr} x_r .$$

Similarly other penalty term becomes

$$B \sum_{t\in T} \left[\sum_{r\in R} b_{tr} x_r + v_t - 1 \right]^2 = B \sum_{t\in T} \left[-\left(\sum_{r\in R} b_{tr} x_r \right)^2 + 2 \sum_{r\in R} b_{tr} x_r \right]$$
$$= -2B \sum_{t\in T} \sum_{r<r'} b_{tr} b_{tr'}\, x_r x_{r'} + 2 \sum_{r\in R} b_{tr} x_r .$$

So we get Coefficients of the coupling term follows:

$$J_{rr'} = 2A \sum_{f\in F} a_{fr} a_{fr'} + 2B \sum_{t\in T} b_{tr} b_{tr'} .$$

and Coefficients of linear term follows:

$$h_r = c_r - \sum_{f\in F} C_f\, a_{fr} - A \sum_{f\in F} a_{fr} - B \sum_{t\in T} b_{tr} .$$

Route cost.

$$E_{\text{cost}} = c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 .$$

Uncovered-flight penalty.

$$E_{\text{miss}} = C_f\,(u_1 + u_2 + u_3 + u_4).$$

Constraint penalty.

$$E_{\text{pen}} = A \sum_{f \in F} \left[\sum_{r \in R} a_{fr} x_r + u_f - 1 \right]^2 + B \sum_{t \in T} \left[\sum_{r \in R} b_{tr} x_r + v_t - 1 \right]^2 .$$

Flight penalty term

$$A\Big[(x_1 + x_3 + u_1 - 1)^2 + (x_2 + x_4 + u_2 - 1)^2 + (x_1 + x_4 + u_3 - 1)^2 + (x_2 + x_3 + u_4 - 1)^2\Big].$$

Aircraft penalty term

$$B\Big[(x_1 + x_3 + v_1 - 1)^2 + (x_2 + x_4 + v_2 - 1)^2\Big].$$

After eliminating slack variables using the substitution $x_i^2 = x_i$ and $u_i^2 = u_i$. The whole expression turns out to be:

$$E(x) = \sum_{r \in R} h_r x_r + \sum_{r < r'} J_{rr'}\, x_r x_{r'} + \text{const.}$$

For example, expanding $(x_1 + x_3 + u_1 - 1)^2$ and substituting $u_1 = 1 - x_1 - x_3$ gives a quadratic coupling term

$$-2x_1 x_3.$$

Similarly we can get all other coupling terms like $\{x_2x_4, x_1x_4, x_2x_3, x_1x_3\}$
The coefficients of the coupling term is given by the following formula:

$$J_{rr'} = 2A \sum_{f \in F} a_{fr} a_{fr'} + 2B \sum_{t \in T} b_{tr} b_{tr'}.$$

and we can compute the following matrix:

$$J_{ij} = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 2A + 2B & 2A \\ 2 & 0 & 0 & 2A & 2A + 2B \\ 3 & 2A + 2B & 2A & 0 & 0 \\ 4 & 2A & 2A + 2B & 0 & 0 \end{array} .$$

The coefficient of linear term is given by

$$h_r = c_r - \sum_{f \in F} C_f\, a_{fr} - A \sum_{f \in F} a_{fr} - B \sum_{t \in T} b_{tr}.$$

For $A = B = C_f = 1$ and $c_r = 0$, we have $\sum_{f \in F} a_{fr} = 2$ for each $r \in \{1, 2, 3, 4\}$ and $\sum_{t \in T} b_{tr} = 1$, hence

$$\mathbf{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{pmatrix} = \begin{pmatrix} -5 \\ -5 \\ -5 \\ -5 \end{pmatrix} .$$

Final QUBO form

$$E(\mathbf{x}) = h_1x_1 + h_2x_2 + h_3x_3 + h_4x_4 + J_{12}x_1x_2 + J_{13}x_1x_3 + J_{14}x_1x_4 + J_{23}x_2x_3 + J_{24}x_2x_4 + J_{34}x_3x_4.$$

$$\begin{aligned} E(\mathbf{x}) &= -5x_1 - 5x_2 - 5x_3 - 5x_4 + 0\,x_1x_2 + 4x_1x_3 + 2x_1x_4 + 2x_2x_3 + 4x_2x_4 + 0\,x_3x_4 \\ &= -5(x_1 + x_2 + x_3 + x_4) + 4x_1x_3 + 2x_1x_4 + 2x_2x_3 + 4x_2x_4. \end{aligned}$$

Convert to Ising Hamiltonian

Use $x_i = \frac{s_i+1}{2}$ with $s_i \in \{-1, +1\}$. Then

$$x_ix_j = \left(\frac{s_i+1}{2}\right)\left(\frac{s_j+1}{2}\right) = \frac{s_is_j + s_i + s_j + 1}{4}.$$

An Ising Hamiltonian can be written as

$$H(\mathbf{s}) = \sum_{i<j} \tilde{J}_{ij}\, s_is_j + \sum_i \tilde{h}_i\, s_i + \text{const}, \qquad \tilde{J}_{ij} = \frac{J_{ij}}{4}.$$

So, the final Hamiltonian comes out to be

$$H(\mathbf{s}) = s_1s_3 + \frac{1}{2}s_1s_4 + \frac{1}{2}s_2s_3 + s_2s_4 - \frac{1}{2}(s_1 + s_2 + s_3 + s_4) + \text{const}.$$

Note: Constants are dropped as they only shift the energy spectrum globally and do not change the location of the minimum.

2 Setting up the QAOA

Since, we successfully setted up the Hamiltonian for our given problem. Now we can proceed to QAOA part.

Step 1: Cost Hamiltonian
To promote the given $H(\mathbf{s})$ to a quantum operator $\hat{H}_C$, we replace the classical spins s_i with Pauli-Z operators $\hat{\sigma}_i^z$:

$$\hat{H}_C = \hat{\sigma}_1^z\hat{\sigma}_3^z + \frac{1}{2}\hat{\sigma}_1^z\hat{\sigma}_4^z + \frac{1}{2}\hat{\sigma}_2^z\hat{\sigma}_3^z + \hat{\sigma}_2^z\hat{\sigma}_4^z - \frac{1}{2}\sum_{i=1}^{4}\hat{\sigma}_i^z.$$

Step 2: Mixer Hamiltonian

In QAOA, the mixer is responsible for inducing quantum tunneling between the computational basis states ($|0\rangle, |1\rangle$) to explore the solution space. The standard choice is a sum of Pauli-X operators acting on each qubit:

$$\hat{H}_M = -\sum_{i=1}^{4}\hat{\sigma}_i^x = -(\hat{\sigma}_1^x + \hat{\sigma}_2^x + \hat{\sigma}_3^x + \hat{\sigma}_4^x)$$

Step 3: Construction of Unitary Operator
we apply these two Hamiltonians alternatively in p layers. For each layer k, we define two unitary gates controlled by parameters γ_k and β_k:

- **Phase Separation Unitary:**

$$U(H_C, \gamma_k) = e^{-i\gamma_k \hat{H}_C} = \exp\left[-i\gamma_k\Big(\hat{\sigma}_1^z\hat{\sigma}_3^z + \frac{1}{2}\hat{\sigma}_1^z\hat{\sigma}_4^z + \frac{1}{2}\hat{\sigma}_2^z\hat{\sigma}_3^z + \hat{\sigma}_2^z\hat{\sigma}_4^z - \frac{1}{2}(\hat{\sigma}_1^z + \hat{\sigma}_2^z + \hat{\sigma}_3^z + \hat{\sigma}_4^z)\Big)\right].$$

  This consists of $R_{ZZ}(\theta)$ gates for the quadratic terms and $R_Z(\theta)$ gates for the linear terms.

- **Mixing Unitary:**

$$U(H_M, \beta_k) = e^{-i\beta_k \hat{H}_M}$$

  This consists of $R_X(2\beta_k)$ gates on every qubit.

3 Circuit Construction

We start with the computational basis state

$$|\psi_0\rangle = |0000\rangle.$$

Applying a Hadamard gate to each qubit prepares a uniform superposition

$$|+\rangle^{\otimes 4} = \frac{1}{2^2}\sum_{z\in\{0,1\}^4}|z\rangle, \qquad |\psi_{\text{init}}\rangle = H^{\otimes 4}|0000\rangle.$$

QAOA state preparation

For depth p, the QAOA ansatz is

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{k=1}^{p} \left(e^{-i\beta_k \hat{H}_M} \, e^{-i\gamma_k \hat{H}_C} \right) |\psi_{\text{init}}\rangle \, .$$

Objective value

We choose parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ to minimize the expected cost

$$\langle \hat{H}_C \rangle(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})| \, \hat{H}_C \, |\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle \, .$$

Now, The QAOA algorithm starts with the ground state of Mixer Hamiltonian $-\sum_{i=1}^{4} \hat{\sigma}_i^x$ and its lowest energy eigenstate is $| + + + + \rangle$. Then it applies alternating unitaries generated by the cost Hamiltonian and mixer Hamiltonian for p layers, with parameters γ_k and β_k to be optimized. Finally, we measure the final state in the computational basis to obtain a candidate solution.

For our problem with 4 qubits, the QAOA circuit with depth $p = 1$ is constructed. At last we are finding lowest expectation value $\langle \psi(\boldsymbol{\gamma}, \boldsymbol{\beta}) | \hat{H}_C | \psi(\boldsymbol{\gamma}, \boldsymbol{\beta}) \rangle$ using the "Cobyla" optimizer from Scipy library.

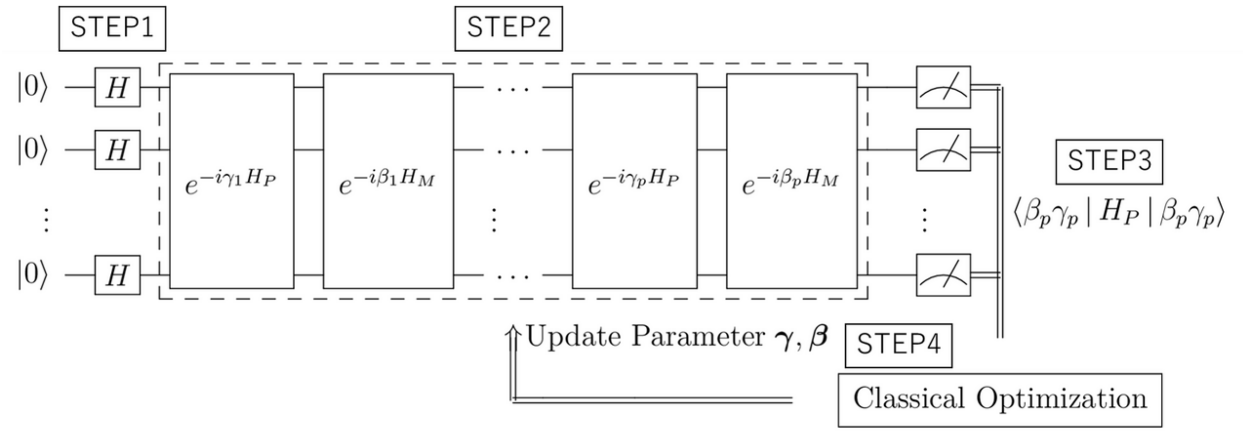

Figure 2: QAOA circuit.

The optimal parameters found are:

$$\gamma = 2.5821, \quad \beta = 0.2662.$$

The following results are obtained after running the circuit with the optimal parameters:

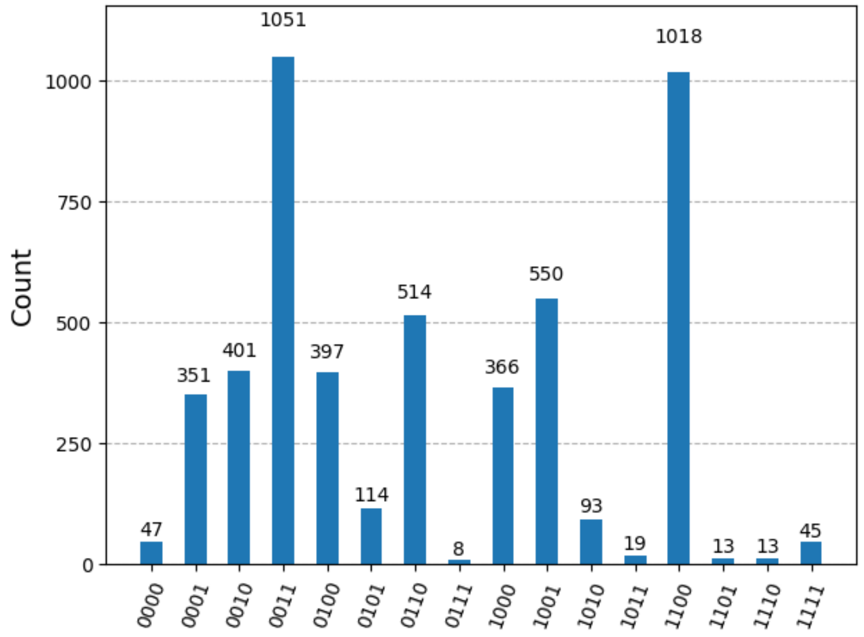

Figure 3: QAOA results.

This shows that there are two optimal solutions:

$$|0011\rangle \quad \text{and} \quad |1100\rangle,$$

which correspond to selecting routes R_1 and R_2, or R_3 and R_4, respectively. Both solutions cover all flights exactly once and satisfy the aircraft assignment constraints.

Note: The problem only focuses on the exact cover constraints and does not include any additional route costs, hence the optimal solutions are those that satisfy the constraints without any cost penalty. In terms of cost, both solutions have the same cost of 0.

4 Generalization of Problem

To implement this generalization, we have to first set up the general n-city problem with m flights and k routes. Then we can construct the incidence matrices a_{fr} and b_{tr} based on the specific flight-route and tail-route relationships. Next, we define the decision variables, constraints, and energy function as done in the simple example. Finally, we convert the energy function to QUBO form, promote it to an Ising Hamiltonian, and set up the QAOA circuit accordingly.

- Set of Tails: $T = \{T_1, T_2, \ldots, T_k\}$
- Set of Flights: $F = \{F_1, F_2, \ldots, F_m\}$
- Set of Routes: $R = \{R_1, R_2, \ldots, R_n\}$
  where $R_1 = \{F_k, F_l, \ldots\}$, $R_2 = \{F_p, F_q, \ldots\}$ and so on.
- Decision variables: $x_r \in \{0, 1\}$ for each $r \in R$.

Now, the incidence matrices a_{fr} and b_{tr} will be:

$$a_{fr} = \begin{array}{c|cccc} & R_1 & R_2 & \cdots & R_n \\ \hline F_1 & a_{11} & a_{12} & \cdots & a_{1n} \\ F_2 & a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ F_m & a_{m1} & a_{m2} & \cdots & a_{mn} \end{array},$$

The dimensionality of this matrix will be $m \times n$ where m is the number of flights and n is the number of routes.

$$b_{tr} = \begin{array}{c|cccc} & R_1 & R_2 & \cdots & R_n \\ \hline T_1 & b_{11} & b_{12} & \cdots & b_{1n} \\ T_2 & b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_k & b_{k1} & b_{k2} & \cdots & b_{kn} \end{array}.$$

The dimensionality of this matrix will be $k \times n$ where k is the number of tails and n is the number of routes.

Now, the constraints equation becomes:

Flight coverage (Exact Cover). Each flight must be covered exactly once:

$$F_1 : a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = 1$$

$$F_2 : a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = 1$$

$$\vdots$$

$$F_m : a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = 1$$

$$\sum_{r\in R} a_{fr}x_r + u_f = 1, \quad \text{for each } f \in F.$$

Aircraft (tail) constraints. Each tail must be assigned to exactly one route:

$$T_1 : b_{11}x_1 + b_{12}x_2 + \cdots + b_{1n}x_n = 1$$

$$T_2 : b_{21}x_1 + b_{22}x_2 + \cdots + b_{2n}x_n = 1$$

$$\vdots$$

$$T_k : b_{k1}x_1 + b_{k2}x_2 + \cdots + b_{kn}x_n = 1$$

$$\sum_{r\in R} b_{tr}x_r + v_t = 1, \quad \text{for each } t \in T.$$

The energy function can be defined as:

$$E(x,u,v) = \underbrace{\sum_{r\in R} c_r x_r}_{\text{true route cost}} + \underbrace{\sum_{f\in F} C_f u_f}_{\text{uncovered-flight penalty (weighted)}}$$

$$\underbrace{+ A\sum_{f\in F}\left(\sum_{r\in R} a_{fr}x_r + u_f - 1\right)^2}_{\text{flight coverage constraint}}$$

$$\underbrace{+ B\sum_{t\in T}\left(\sum_{r\in R} b_{tr}x_r + v_t - 1\right)^2}_{\text{tail assignment constraint}}$$

After Combining all the term and removing slack variables using the substitution $u_f = 1 - \sum_{r\in R} a_{fr}x_r$ and $v_t = 1 - \sum_{r\in R} b_{tr}x_r$, we get coupling, linear and constant terms like:

$$E(x) = \sum_{r\in R} h_r x_r + \sum_{r<r'} J_{rr'}\, x_r x_{r'} + \text{const.}$$

and its coefficient is given by:

$$J_{rr'} = 2A\sum_{f\in F} a_{fr}a_{fr'} + 2B\sum_{t\in T} b_{tr}b_{tr'},$$

$$h_r = c_r - \sum_{f\in F} C_f\, a_{fr} - A\sum_{f\in F} a_{fr} - B\sum_{t\in T} b_{tr}.$$

Finally, we can convert this to an Ising Hamiltonian and set up the QAOA circuit as done in the simple example, but now with n qubits corresponding to the n routes. The cost Hamiltonian will be constructed based on the new coefficients $J_{rr'}$ and h_r, and the mixer Hamiltonian remains the same as a sum of Pauli-X operators on all qubits.

5 Code Implementation

The `AirlineQAOA` class is implemented in Python using the Qiskit library. It includes methods for constructing the cost Hamiltonian, mixer Hamiltonian, and the QAOA circuit. The class also has a method to optimize the parameters γ and β using a classical optimizer to minimize the expected value of the cost Hamiltonian.

Now, let's see the function definitions and their purpose in the code:

`__init__(self, afr_matrix, btr_matrix, route_costs, penalty_A, penalty_B, flight_penalties)`

Initializes the problem instance by mapping airline constraints to a QUBO framework.

- **Parameters:** Sets the flight-route incidence matrix (afr), tail-route incidence matrix (btr), and associated costs/penalties.
- **Attributes:** Calculates dimensions for flights (m), routes (n), and tails (k).
- **Logic:** Invokes `_compute_qubo_coefficients()` to generate the Hamiltonian parameters J and h.

`_compute_qubo_coefficients(self)`

Calculates the coupling matrix J and linear vector h for the Quadratic Unconstrained Binary Optimization (QUBO) form.

- **Quadratic Terms (J):** Computes $J_{rr'} = 2A\sum_f(a_{fr}a_{fr'}) + 2B\sum_t(b_{tr}b_{tr'})$ to penalize overlapping routes.
- **Linear Terms (h):** Computes $h_r = c_r - \sum_f(C_f a_{fr}) - A\sum_f a_{fr} - B\sum_t b_{tr}$.
- **Returns:** A symmetric matrix J and a vector h.

`problem_summary(self)`

Prints an overview of the problem setup, including the incidence matrices, penalty weights, and the resulting QUBO coefficients.

`build_qaoa(self, p_layers)`

Constructs the parameterized Qiskit quantum circuit for the QAOA algorithm.

- **State Prep:** Applies Hadamard gates (H) to all qubits to create a uniform superposition $|+\rangle^n$.

- **Structure:** Iteratively adds p layers consisting of a Cost Unitary followed by a Mixer Unitary.
- **Parameters:** Defines `Parameter` objects for γ and β for each layer.

`_apply_cost_unitary(self, qc, qr, gamma)`

Applies the phase separation operator $U(C, \gamma) = e^{-i\gamma H_C}$.

- **Quadratic Gates:** Implements RZZ gates between qubits r and r' proportional to $2\gamma J_{rr'}$.
- **Linear Gates:** Implements RZ gates on qubit r proportional to $2\gamma h_r$.

`_apply_mixer_unitary(self, qc, qr, beta)`

Applies the mixing operator $U(B, \beta) = e^{-i\beta H_M}$.

- **Logic:** Applies RX gates to every qubit with a rotation angle of 2β.

`compute_energy(self, bitstring)`

Calculates the scalar energy value of a specific classical bitstring.

- **Calculation:** Computes $E = \sum h_r x_r + \sum J_{rr'} x_r x_{r'}$.
- **Returns:** A float representing the cost of the configuration.

`compute_expectation(self, counts)`

Calculates the average energy (expectation value) for a set of measurement results.

- **Logic:** Sums the energy of each measured bitstring weighted by its probability of occurrence ($count/total\_shots$).

`verify_solution(self, bitstring)`

The function determines if the binary string (solution) obeys the two "hard" constraints of the airline routing problem:

- **Flight Coverage:** It checks if every flight in the schedule is covered exactly once by the selected routes.
- **Tail Assignment:** It checks if every tail is assigned exactly once by the selected routes.
- **Returns:** A boolean `valid` and a dictionary of detailed metrics.

To perform these validations, the function converts the bitstring into a vector x and uses matrix-vector multiplication:

$$\text{flight\_coverage} = a_{fr} \cdot x, \quad \text{tail\_assignment} = b_{tr} \cdot x.$$

The solution is considered valid only if the resulting values for both operations are all 1s.

`optimize(self, p_layers, initial_params, shots, method)`

Executes the classical-quantum optimization loop.

- **Objective:** Uses `scipy.optimize.minimize` to find optimal γ and β values that minimize the expectation value.
- **Execution:** Binds optimal parameters to the circuit and runs a high-shot final simulation to gather results.

`analyze_results(self, counts, top_k, valid_sol)`

Processes the raw measurement data into a readable format.

- **Logic:** Sorts bitstrings by frequency and prints their validity, energy, and selected route indices.

6 Implementation workflow

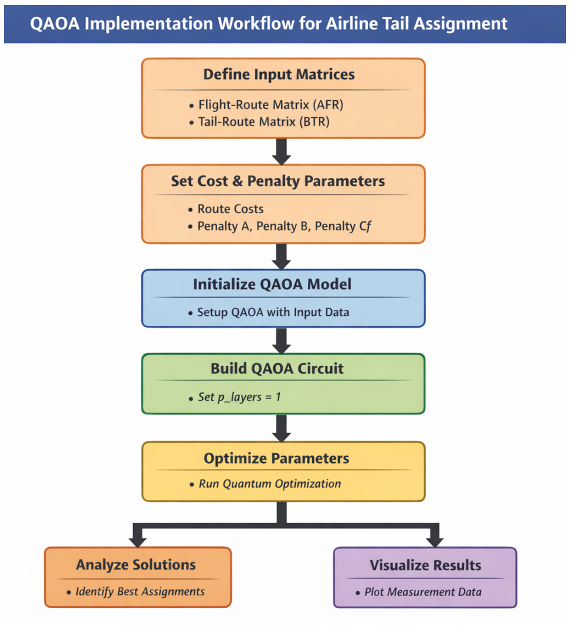
QAOA Implementation Workflow for Airline Tail Assignment
Define Input Matrices
• Flight-Route Matrix (AFR)
• Tail-Route Matrix (BTR)
Set Cost & Penalty Parameters
• Route Costs
• Penalty A, Penalty B, Penalty Cf
Initialize QAOA Model
• Setup QAOA with Input Data
Build QAOA Circuit
• Set p_layers = 1
Optimize Parameters
• Run Quantum Optimization
Analyze Solutions
• Identify Best Assignments
Visualize Results
• Plot Measurement Data

Figure 4: Implementation workflow.

6.1 Example usage

This is the example usage of the above 4 qubit problem.

```python
from airlines_qaoa import AirlineQAOA
import numpy as np
from qiskit.visualization import plot_histogram

afr = np.array([
        [1, 0, 1, 0],  # F1 in R1, R3
        [0, 1, 0, 1],  # F2 in R2, R4
        [1, 0, 0, 1],  # F3 in R1, R4
        [0, 1, 1, 0]   # F4 in R2, R3
    ])

btr = np.array([
        [1, 0, 1, 0],  # Tail A assigned to R1, R3
        [0, 1, 0, 1]   # Tail B assigned to R2, R4
    ])

qaoa = AirlineQAOA(
        afr_matrix=afr,
        btr_matrix=btr,
        route_costs=np.zeros(4),
        penalty_A=1.0,
        penalty_B=1.0,
        flight_penalties=np.ones(4),
    )

qaoa.problem_summary()

qaoa.build_qaoa(p_layers=1)

results = qaoa.optimize(p_layers=1, initial_params=[1.0, 1.0], shots
    =1024)
print(results['counts'])

solutions = qaoa.analyze_results(results['counts'], top_k=5)

plot_histogram(results['counts'])
```