Optimizing Clinical Trials Based on the Mayo Clinic Dataset

Team : NeuroCat

Niloy Kumar Mondal

Bangladesh University of Engineering and Technology
github.com/QuitttCat
nkm2105044@gmail.com
+8801792023909

H. M. Shadman Tabib
Bangladesh University of Engineering and Technology
github.com/shadmantabib
+880-1880-198766

November 9, 2025

Contents

1 Problem formulation and QUBO instance creation

1.1 Mathematical breakdown

First term of discrepancies

We begin with the definition of the mean-difference term:

$$\Delta\mu_s = \frac{1}{n}\sum_{i=1}^{n} w_{is}(x_{i1} - x_{i2}),$$

where

$$x_{i1} - x_{i2} = 2y_i - 1, \qquad y_i \in \{0, 1\}.$$

Substituting:

$$\Delta\mu_s = \frac{2}{n}\sum_{i=1}^{n} w_{is}y_i - \frac{1}{n}\sum_{i=1}^{n} w_{is}.$$

Define

$$c_s = \frac{1}{n}\sum_{i=1}^{n} w_{is}.$$

Then

$$\boxed{\Delta\mu_s = \frac{2}{n}\sum_{i=1}^{n} w_{is}y_i - c_s.}$$

We want to minimize

$$\min_{y\in\{0,1\}^n} \sum_{s=1}^{3} (\Delta\mu_s)^2.$$

Expanding $(\Delta\mu_s)^2$

$$(\Delta\mu_s)^2 = \left(\frac{2}{n}\sum_i w_{is}y_i - c_s\right)^2 = \frac{4}{n^2}\left(\sum_i w_{is}y_i\right)^2 - \frac{4c_s}{n}\sum_i w_{is}y_i + c_s^2.$$

Expanding the squared sum:

$$\left(\sum_i w_{is}y_i\right)^2 = \sum_i\sum_j w_{is}w_{js}y_iy_j.$$

Therefore:

$$\boxed{(\Delta\mu_s)^2 = \frac{4}{n^2}\sum_i\sum_j w_{is}w_{js}y_iy_j - \frac{4c_s}{n}\sum_i w_{is}y_i + c_s^2.}$$

Separating Diagonal and Off-Diagonal Terms

Split the double sum:

$$\sum_i \sum_j w_{is} w_{js} y_i y_j = \sum_i w_{is}^2 y_i^2 + \sum_{i<j} 2 w_{is} w_{js} y_i y_j.$$

Since $y_i^2 = y_i$:

$$(\Delta\mu_s)^2 = \frac{4}{n^2} \left[\sum_i w_{is}^2 y_i + \sum_{i<j} 2 w_{is} w_{js} y_i y_j \right] - \frac{4c_s}{n} \sum_i w_{is} y_i + c_s^2.$$

Identifying the QUBO Coefficients

From the above, the QUBO entries for a fixed s are:

$$Q_{ii}^{(s)} = \frac{4}{n^2} w_{is}^2 - \frac{4c_s}{n} w_{is}, \qquad Q_{ij}^{(s)} = \frac{8}{n^2} w_{is} w_{js}, \quad i < j.$$

Summing over $s = 1, 2, 3$:

$$\boxed{Q_{ii} = \sum_{s=1}^{3} \left(\frac{4}{n^2} w_{is}^2 - \frac{4c_s}{n} w_{is} \right), \qquad Q_{ij} = \sum_{s=1}^{3} \frac{8}{n^2} w_{is} w_{js}.}$$

The constant term $\sum_s c_s^2$ can be ignored for minimization.

Scaled Simplification

Dropping the global $1/n^2$ and $1/n$ factors (which only scale the energy):

$$\boxed{Q_{ii} \propto \sum_{s=1}^{3} 4(w_{is}^2 - w_{is} w_s^{\text{sum}}), \qquad Q_{ij} \propto \sum_{s=1}^{3} 8 w_{is} w_{js},}$$

where $w_s^{\text{sum}} = \sum_r w_{rs}$.

Python / NumPy Implementation

```
import numpy as np
def build_Q1(w: np.ndarray) -> np.ndarray:
    #takes feature matrix as parameter,returns first discrepency
       term
    n = w.shape[0]  #n denotes the total number of patient
    w_sum = np.sum(w, axis=0)  # column sums
    Q1 = np.zeros((n, n))

    for i in range(n):
        # Diagonal part
        for s in range(3):
            Q1[i, i] += 4 * (w[i, s]**2 - w[i, s] * w_sum[s])
```

```
        # Off-diagonal part
        for j in range(i + 1, n):
            for s in range(3):
                Q1[i, j] += 8 * w[i, s] * w[j, s]

    return Q1
```

Second term of discrepancy

We now consider the diagonal part of the covariance discrepancy term:

$$\rho \sum_{s=1}^{3} |\Delta\sigma_{ss}|.$$

For the QUBO formulation, we replace the absolute value with a square:

$$\rho \sum_{s=1}^{3} (\Delta\sigma_{ss})^2.$$

Each diagonal element is defined as

$$\Delta\sigma_{ss} = \frac{1}{n} \sum_{i=1}^{n} w_{is}^2 (x_{i1} - x_{i2}).$$

Using binary variables $y_i \in \{0, 1\}$ with $(x_{i1} - x_{i2}) = 2y_i - 1$:

$$\Delta\sigma_{ss} = \frac{2}{n} \sum_{i=1}^{n} w_{is}^2 y_i - \frac{1}{n} \sum_{i=1}^{n} w_{is}^2.$$

Let

$$u_{is} := w_{is}^2, \qquad c'_s := \frac{1}{n} \sum_{i=1}^{n} u_{is}.$$

Then:

$$(\Delta\sigma_{ss})^2 = \left(\frac{2}{n} \sum_i u_{is} y_i - c'_s \right)^2 = \frac{4}{n^2} \sum_i \sum_j u_{is} u_{js} y_i y_j - \frac{4c'_s}{n} \sum_i u_{is} y_i + (c'_s)^2.$$

Ignoring constants and scaling (since they do not affect the minimizer):

$$E_2^{\text{diag}}(y) = \rho \sum_{s=1}^{3} \left[\sum_i 4(u_{is}^2 - u_{is} u_{\Sigma s}) y_i + \sum_{i<j} 8 u_{is} u_{js} y_i y_j \right],$$

where

$$u_{\Sigma s} = \sum_{r=1}^{n} u_{rs} = \sum_{r=1}^{n} w_{rs}^2.$$

Hence, the QUBO coefficients for this term are:

$$\boxed{Q_{ii}^{(2)} = \rho \sum_{s=1}^{3} 4(u_{is}^2 - u_{is} u_{\Sigma s}), \qquad Q_{ij}^{(2)} = \rho \sum_{s=1}^{3} 8 u_{is} u_{js}, \ i < j.}$$

Python / NumPy Implementation

```
import numpy as np

def build_Q2(w: np.ndarray, rho: float = 0.5) -> np.ndarray:

    n = w.shape[0]                      # number of patients
    Q2 = np.zeros((n, n))               # initialize QUBO matrix

    u = w**2                            # u[i, s] = w[i, s]^2
    u_sum = np.sum(u, axis=0)

    for i in range(n):
        Q2[i, i] = rho * np.sum(4 * (u[i]**2 - u[i] * u_sum))


    for i in range(n):
        for j in range(i + 1, n):
            Q2[i, j] = rho * np.sum(8 * u[i] * u[j])



    return Q2
```

Third term of discrepancy

We now consider the off-diagonal part of the covariance discrepancy:

$$2\rho \sum_{s=1}^{3} \sum_{s'=s+1}^{3} |\Delta\sigma_{ss'}| .$$

For the QUBO formulation, we replace the absolute value with a square:

$$2\rho \sum_{s=1}^{3} \sum_{s'=s+1}^{3} (\Delta\sigma_{ss'})^2 .$$

Each off-diagonal element is defined as

$$\Delta\sigma_{ss'} = \frac{1}{n} \sum_{i=1}^{n} w_{is} w_{is'} (x_{i1} - x_{i2}), \qquad s' > s.$$

Using binary variables $y_i \in \{0, 1\}$ with $(x_{i1} - x_{i2}) = 2y_i - 1$:

$$\Delta\sigma_{ss'} = \frac{2}{n} \sum_{i=1}^{n} w_{is} w_{is'} y_i - \frac{1}{n} \sum_{i=1}^{n} w_{is} w_{is'}.$$

For each pair (s, s') with $s' < s + 1$, define

$$v_i^{(ss')} := w_{is} w_{is'}, \qquad c_{ss'} := \frac{1}{n} \sum_{i=1}^{n} v_i^{(ss')}.$$

Then

$$(\Delta\sigma_{ss'})^2 = \left(\frac{2}{n} \sum_i v_i^{(ss')} y_i - c_{ss'} \right)^2 = \frac{4}{n^2} \sum_i \sum_j v_i^{(ss')} v_j^{(ss')} y_i y_j - \frac{4c_{ss'}}{n} \sum_i v_i^{(ss')} y_i + c_{ss'}^2.$$

Separating diagonal and off-diagonal indices:

$$(\Delta\sigma_{ss'})^2 = \sum_i \left(\frac{4}{n^2} \left(v_i^{(ss')} \right)^2 - \frac{4c_{ss'}}{n} v_i^{(ss')} \right) y_i + \sum_{i<j} \frac{8}{n^2} v_i^{(ss')} v_j^{(ss')} y_i y_j + c_{ss'}^2.$$

Ignoring constants and common scaling (which do not affect the minimizer), the off-diagonal covariance contribution is

$$E_3(y) = 2\rho \sum_{s=1}^{3} \sum_{s'=s+1}^{3} \left[\sum_i 4\big((v_i^{(ss')})^2 - v_i^{(ss')} v_\Sigma^{(ss')} \big) y_i + \sum_{i<j} 8 v_i^{(ss')} v_j^{(ss')} y_i y_j \right],$$

where

$$v_\Sigma^{(ss')} = \sum_{r=1}^{n} v_r^{(ss')} = \sum_{r=1}^{n} w_{rs} w_{rs'}.$$

Hence, the QUBO coefficients for this third term are:

$$\boxed{Q_{ii}^{(3)} = 2\rho \sum_{s=1}^{3} \sum_{s'=s+1}^{3} 4\big((v_i^{(ss')})^2 - v_i^{(ss')} v_\Sigma^{(ss')} \big),}$$

$$\boxed{Q_{ij}^{(3)} = 2\rho \sum_{s=1}^{3} \sum_{s'=s+1}^{3} 8\, v_i^{(ss')} v_j^{(ss')}, \qquad i < j.}$$

Python / NumPy Implementation

```
import numpy as np

def build_Q3(w: np.ndarray, rho: float = 0.5) -> np.ndarray:

    n_samples, n_features = w.shape
    Q3 = np.zeros((n_samples, n_samples))

    # building all v_iˆ(ss') for s' > s and their sums over i
    pair_list = []
    for s in range(n_features):
        for sp in range(s + 1, n_features):
            pair_list.append((s, sp))

    n_pairs = len(pair_list)
    V = np.zeros((n_samples, n_pairs))
    for idx, (s, sp) in enumerate(pair_list):
        V[:, idx] = w[:, s] * w[:, sp]

    v_total = np.sum(V, axis=0)  # v_Sigmaˆ(ss') for each pair

    # diagonal entries
    for i in range(n_samples):
        v_i = V[i]                          # all v_iˆ(ss')
        Q3[i, i] = 2.0 * rho * np.sum(
            4.0 * (v_i**2 - v_i * v_total)
        )

    # off-diagonal entries
    for i in range(n_samples):
        for j in range(i + 1, n_samples):
            v_i = V[i]
            v_j = V[j]
            Q3[i, j] = 2.0 * rho * np.sum(
                8.0 * v_i * v_j
            )

    return Q3
```

Constraints Enforcement

Constraint: Each patient assigned to exactly one group

This constraint ensures that every patient i belongs to exactly one of the two groups:

$$x_{i1} + x_{i2} = 1, \qquad \forall i \in \{1, \ldots, n\}.$$

In our QUBO formulation, we introduced a single binary decision variable $y_i \in \{0, 1\}$ for each patient, representing the group assignment:

$$y_i = \begin{cases} 1, & \text{if patient } i \text{ is assigned to group 1,} \\ 0, & \text{if patient } i \text{ is assigned to group 2.} \end{cases}$$

We relate these variables to the original binary indicators as

$$x_{i1} = y_i, \qquad x_{i2} = 1 - y_i.$$

Substituting these definitions, we find that

$$x_{i1} + x_{i2} = y_i + (1 - y_i) = 1,$$

which is identically satisfied for all i regardless of the optimization outcome.

Hence, this constraint is inherently guaranteed by the binary encoding and does not require any additional penalty term in the QUBO formulation.

The one-group-per-patient constraint is automatically enforced by construction.

Constraint: Number of patients in each group

Groups 1 and 2 need to have the same number of patients, $n/2$. This requirement can be written in terms of the original variables x_{ip} as

$$\sum_{i=1}^{n} x_{ip} = \frac{n}{2}, \qquad \forall p \in \{1, 2\}.$$

Using the encoding

$$x_{i1} = y_i, \qquad x_{i2} = 1 - y_i, \qquad y_i \in \{0, 1\},$$

it is sufficient to enforce the balance in one group, for example group 1:

$$\sum_{i=1}^{n} y_i = \frac{n}{2}.$$

QUBO penalty for the size constraint

We introduce a quadratic penalty with weight $\lambda_{\text{size}} > 0$:

$$E_{\text{size}}(y) = \lambda_{\text{size}} \left(\sum_{i=1}^{n} y_i - \frac{n}{2} \right)^2.$$

Expanding the square,

$$\left(\sum_i y_i - \frac{n}{2} \right)^2 = \left(\sum_i y_i \right)^2 - n \sum_i y_i + \frac{n^2}{4}.$$

The squared sum can be written as

$$\left(\sum_i y_i \right)^2 = \sum_i y_i^2 + 2 \sum_{i<j} y_i y_j = \sum_i y_i + 2 \sum_{i<j} y_i y_j,$$

since $y_i^2 = y_i$ for binary variables. Therefore,

$$\left(\sum_i y_i - \frac{n}{2} \right)^2 = (1 - n) \sum_i y_i + 2 \sum_{i<j} y_i y_j + \frac{n^2}{4}.$$

Ignoring the constant $\frac{n^2}{4}$, the contribution to the QUBO energy is

$$E_{\text{size}}(y) = \lambda_{\text{size}} \left[(1 - n) \sum_i y_i + 2 \sum_{i<j} y_i y_j \right] + \text{const.}$$

Hence the QUBO coefficients for this constraint are

$$\boxed{Q_{ii}^{\text{size}} = \lambda_{\text{size}}(1 - n), \qquad Q_{ij}^{\text{size}} = 2\lambda_{\text{size}}, \quad i < j.}$$

Python / NumPy Implementation

```python
import numpy as np

def add_group_size_penalty(Q: np.ndarray, lam_size: float = 3.0)
   -> np.ndarray:

    Q_new = Q.copy()
    n = Q.shape[0]

    # Diagonal contributions: lam_size * (1 - n)
    for i in range(n):
        Q_new[i, i] += lam_size * (1.0 - n)

    # Off-diagonal contributions: 2 * lam_size for i < j
    for i in range(n):
        for j in range(i + 1, n):
            Q_new[i, j] += 2.0 * lam_size

    return Q_new
```

Symmetry redundancy and why the constraint can be omitted

The symmetry redundancy constraint is introduced to break the permutation symmetry between group labels. It is written in the original x-variables as

$$x_{12} = 0.$$

Recall that x_{ip} indicates whether patient i is assigned to group $p \in \{1, 2\}$. Fixing $x_{12} = 0$ means that patient 1 is not in group 2.

Rewriting the symmetry constraint in the y-encoding

In our formulation we use a single binary variable $y_i \in \{0, 1\}$ per patient and set

$$x_{i1} = y_i, \qquad x_{i2} = 1 - y_i.$$

For patient 1 we therefore have

$$x_{11} = y_1, \qquad x_{12} = 1 - y_1.$$

The symmetry constraint becomes

$$x_{12} = 0 \quad \Longleftrightarrow \quad 1 - y_1 = 0 \quad \Longleftrightarrow \quad y_1 = 1.$$

Thus, in the y-variables the symmetry constraint simply fixes the first patient to group 1:

$$y_1 = 1.$$

Symmetry transformation

Consider the transformation that swaps the group labels 1 and 2:

$$x'_{i1} = x_{i2}, \qquad x'_{i2} = x_{i1} \qquad \forall i.$$

In terms of y_i, this corresponds to

$$y'_i = 1 - y_i, \qquad \forall i,$$

since

$$x'_{i1} = x_{i2} = 1 - y_i = y'_i, \qquad x'_{i2} = x_{i1} = y_i = 1 - y'_i.$$

We now show that

1. all constraints (except $y_1 = 1$) are invariant under $y \mapsto 1 - y$, and 2. the total QUBO energy is also invariant.

Group-size constraint. The size constraint in y-variables is

$$\sum_{i=1}^{n} y_i = \frac{n}{2}.$$

Under $y'_i = 1 - y_i$ we have

$$\sum_{i=1}^{n} y'_i = \sum_{i=1}^{n} (1 - y_i) = n - \sum_{i=1}^{n} y_i.$$

If y satisfies the size constraint, then

$$\sum_{i=1}^{n} y_i = \frac{n}{2} \quad \Longrightarrow \quad \sum_{i=1}^{n} y'_i = n - \frac{n}{2} = \frac{n}{2},$$

so the size constraint is preserved.

One-group-per-patient constraint. By construction,

$$x_{i1} + x_{i2} = y_i + (1 - y_i) = 1,$$

for every i, and the same holds for x'_{i1}, x'_{i2}. Hence this constraint is also invariant under the swap.

Discrepancy terms. All three discrepancy terms E_1, E_2, E_3 are built from differences

$$x_{i1} - x_{i2} = 2y_i - 1.$$

Under the swap,

$$x'_{i1} - x'_{i2} = x_{i2} - x_{i1} = -(x_{i1} - x_{i2}) = -(2y_i - 1) = 2y'_i - 1.$$

Thus, each $\Delta\mu_s$ and each $\Delta\sigma_{ss'}$ changes sign, but the objective contains only squared terms:

$$(\Delta\mu'_s)^2 = (\Delta\mu_s)^2, \qquad (\Delta\sigma'_{ss'})^2 = (\Delta\sigma_{ss'})^2.$$

Therefore,

$$E_{\text{disc}}(y') = E_{\text{disc}}(y).$$

Size-penalty term. The group-size penalty was chosen as

$$E_{\text{size}}(y) = \lambda_{\text{size}} \left(\sum_i y_i - \frac{n}{2} \right)^2 .$$

Since $\sum_i y'_i = n - \sum_i y_i$, we obtain

$$\sum_i y'_i - \frac{n}{2} = n - \sum_i y_i - \frac{n}{2} = - \left(\sum_i y_i - \frac{n}{2} \right),$$

so

$$E_{\text{size}}(y') = \lambda_{\text{size}} \left(\sum_i y'_i - \frac{n}{2} \right)^2 = \lambda_{\text{size}} \left(\sum_i y_i - \frac{n}{2} \right)^2 = E_{\text{size}}(y).$$

Combining these observations, the total energy

$$E_{\text{total}}(y) = E_{\text{disc}}(y) + E_{\text{size}}(y)$$

satisfies

$$E_{\text{total}}(1 - y) = E_{\text{total}}(y)$$

for every feasible y (i.e., satisfying the size constraint).

Redundancy of the symmetry constraint

Let $\mathcal{F}$ denote the set of feasible assignments satisfying all constraints *except* $y_1 = 1$. For any $y \in \mathcal{F}$, the transformed assignment $1 - y$ is also in $\mathcal{F}$ and has the same energy:

$$y \in \mathcal{F} \quad \Longrightarrow \quad 1 - y \in \mathcal{F}, \quad E_{\text{total}}(1 - y) = E_{\text{total}}(y).$$

Let $y^\star$ be a minimizer of E_{total} over $\mathcal{F}$. If $(y^\star)_1 = 1$, then $y^\star$ already satisfies the symmetry constraint. If $(y^\star)_1 = 0$, then the transformed solution $\tilde{y} = 1 - y^\star$ satisfies

$$\tilde{y}_1 = 1, \qquad E_{\text{total}}(\tilde{y}) = E_{\text{total}}(y^\star),$$

so there always exists an optimal solution with $y_1 = 1$.

Consequently, adding the explicit penalty enforcing $y_1 = 1$ (equivalently $x_{12} = 0$) does not change the minimum energy value; it only removes one of two energetically equivalent labelings. For this reason, the symmetry redundancy constraint can be safely omitted without affecting the optimization objective:

$$\min_{y \in \mathcal{F}} E_{\text{total}}(y) = \min_{\substack{y \in \mathcal{F} \\ y_1 = 1}} E_{\text{total}}(y).$$

2 Qubo Solver Pipeline

2.1 Quantum Solver Implementation

The pure quantum solution is implemented using the D-Wave Ocean SDK, specifically utilizing the Quantum Processing Unit (QPU) directly. The Quadratic Unconstrained Binary Optimization (QUBO) matrix Q is first converted into a Binary Quadratic Model (BQM). To address the mismatch between the logical problem graph (fully connected in many generic QUBOs) and the physical hardware graph of the QPU (e.g., Pegasus or Zephyr topology), we employ the `EmbeddingComposite` layer.

This composite acts as a wrapper around the base `DWaveSampler`. It automatically handles 'minor embedding,' the process of mapping logical variables to chains of physical qubits. The solver performs quantum annealing N times (defined by the `num_reads` parameter, set to 100 in our initial runs) to generate a distribution of potential solutions. The lowest energy state found across these reads is selected as the candidate solution. This approach is best suited for problems that fit entirely within the QPU's working graph and allows for fine-grained control over annealing parameters such as chain strength.

2.2 Hybrid Solver Implementation

For larger problem instances or those with complex connectivity that may exceed the efficient embedding capabilities of the pure QPU, we utilize D-Wave's Leap Hybrid Solver via the `LeapHybridSampler`. This solver leverages a portfolio of both classical algorithms and quantum annealing.

The hybrid approach abstracts the low-level hardware details. It accepts the BQM and automatically utilizes classical high-performance computing resources to decompose the large problem into smaller sub-problems. These sub-problems are then sent to the QPU for rapid sampling. The classical resources subsequently recombine these quantum-sampled components to construct a high-quality global solution. This method generally allows for handling significantly larger numbers of variables than pure quantum annealing and often finds competitive solutions faster for complex standard industry problems, as it avoids the overhead of manual embedding and parameter tuning.

2.3 Code Workthrough

Open Google Colab Notebook

This contains QUBO implementation using python ,some classical overview and code needed to run on D-wave hardware

2.4 Result Analysis

Unfortunately, D-wave cannot be accessed from Bangladesh. So we could not access hardware but set up the full pipeline so that anyone can run it.

3 A NISQ Hardware-Aware QAOA Pipeline

The pipeline is broken down into the following key stages:

3.1 Problem Formulation and Hamiltonian

The problem is first formulated as a Quadratic Unconstrained Binary Optimization (QUBO), a standard format for optimization. This is done using functions `build_Q1`, `build_Q2`, and `build_Q3`, which translate the core objective—minimizing statistical discrepancies between two patient groups—into a quadratic cost matrix Q. A penalty term is added via `add_group_size_penalty` to enforce the real-world constraint that the two groups must be of equal size. This final QUBO matrix is then converted into a Pauli Hamiltonian (H_C) using `build_pauli_hamiltonian_from_Q`. This Hamiltonian is essential because its ground state directly represents the optimal solution to the problem, making it solvable by QAOA.

3.2 Hardware-Aware Layout Selection

A critical and non-standard feature of this pipeline is the proactive selection of a specific, high-quality qubit layout on the target hardware.

1. **Qubit Scoring:** Instead of letting the transpiler choose any 100 qubits, the script first queries the backend for calibration data (T1, T2, readout error) and computes a heuristic "quality score" for every qubit.
2. **Linear Chain Search(DFS on coupling map):**
   - We find best physical qubits first with low decoherence
   - We need a XY mixer or H_{ring} ansatz best fitted for this kinda problem
   - XY mixer needs connection to ith and i+1 th qubits.But they may be physically disconnected
   - So we search on coupling map for chaining so that we can efficiently and correctly apply this method
3. **Layout Pinning:** The resulting list of 100 qubits (e.g., `connected_100`) is saved and used as a mandatory `initial_layout` for the transpiler. This forces the quantum circuit to be mapped to this specific, pre-selected chain.

3.3 Co-Designed QAOA Ansatz

The `QAOAAnsatz` is then constructed not from standard components, but from three parts that are co-designed to work with the selected hardware layout and problem structure.

1. **Initial State (Warm-Start):** The algorithm does not start in the standard superposition state $|+\rangle^{\otimes n}$. Instead, it uses a "warm-start" by first running a classical heuristic (`get_warm_start_closest_pairs`) to find a good-but-suboptimal solution. This classical solution is encoded into the initial state of the quantum circuit (e.g., $|1001...01\rangle$), giving the algorithm a significant head start. The run time of the classical closest pair heuristic is $O(n^2)$, where n is the number of patients, as it involves

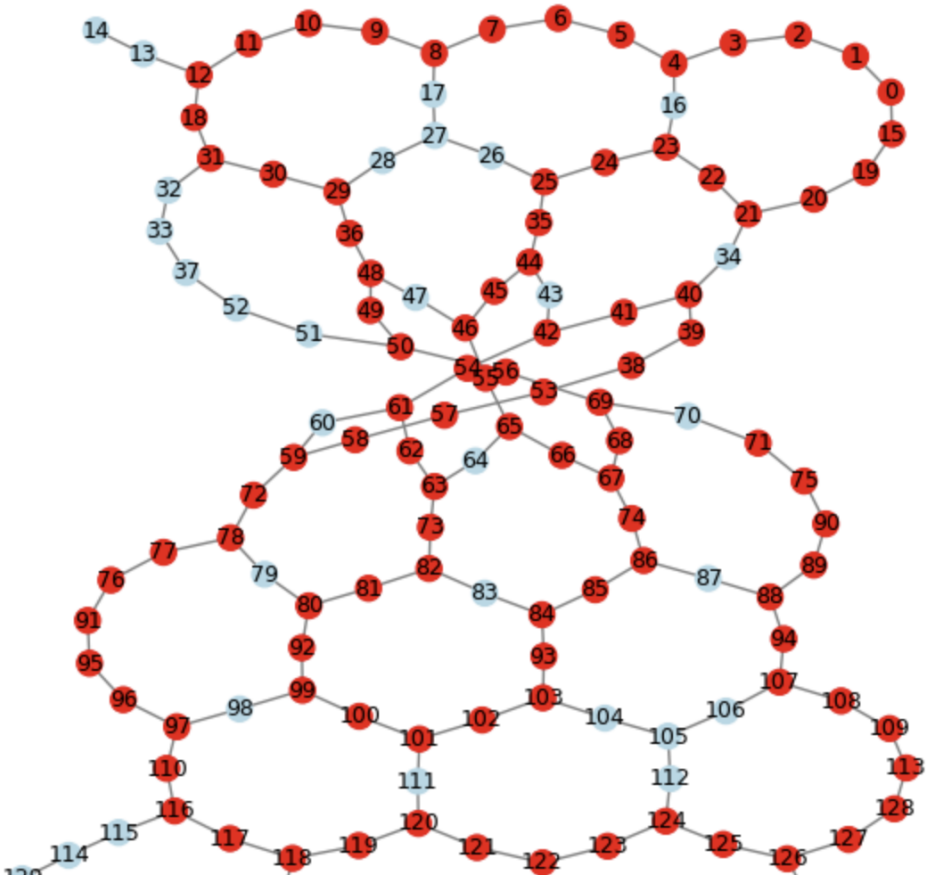

Figure 1: DFS on coupling map

computing pairwise distances between all patients. This step, though computationally intensive, provides an efficient starting point for the quantum optimization and reduces the search space for the quantum algorithm.

2. **Cost Operator:** This is the H_C derived in the first stage.

3. **Mixer Operator (Topology-Aware):** The standard X-mixer ($H_M = \sum_i X_i$) is replaced with a custom `XY` mixer created by `create_xy_linear_mixer`. This mixer is specifically designed to match the 100-qubit linear chain, containing only XX and YY terms that couple adjacent qubits $(i, i+1)$ in the chain. This is crucial because these two-qubit interactions map directly to the hardware's native connectivity, eliminating the need for costly SWAP gates.

3.4 Transpilation and Execution Pipeline

The final stage brings all previous components together for execution.

1. **Layout-Aware Transpilation:** The `generate_preset_pass_manager` is explicitly given the `initial_layout = connected_100`. This instructs the transpiler to optimize the circuit *under the constraint* that it must run on this exact linear chain. Because the mixer and layout are co-designed, the transpilation process is highly efficient and results in a circuit with minimal overhead.

2. **Hybrid Optimization Loop:** The classical optimizer (`scipy.optimize.minimize`) repeatedly calls the `cost_func_estimator`. This function packages the transpiled circuit and current parameters (γ, β) and sends them to the `EstimatorV2` primitive on the backend to measure the expected cost $\langle\psi(\gamma,\beta)|H_C|\psi(\gamma,\beta)\rangle$.

3. **Final Sampling:** Once the optimizer finds the optimal parameters, the `SamplerV2` primitive is used to run the final, optimized circuit to generate bitstring samples, which represent the candidate solutions to the original problem.

3.5 Code Walkthrough

Open Google Colab Notebook

Inspired by IBM max cut utility scale formulation

3.6 Result analysis

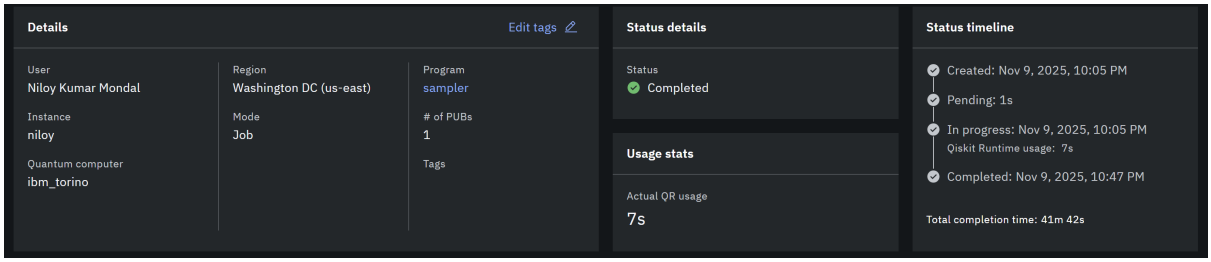

Figure 2: IBM runtime : Torino , number of shots 2, time 7s

We need IBM credits to run this. This will run completely in 30 minutes approximated by IBM on heron 2.

4 A feasible solution : Classical Batch with Quantum Exhaustive Search

4.1 Feasibility benchmarking as stated by IBM

This tutorial demonstrated how to solve an optimization problem with a quantum computer using the Qiskit pattern framework. The demonstration included a utility-scale example, with circuit sizes that cannot be exactly simulated classically. Currently, quantum computers do not outperform classical computers for combinatorial optimization because of noise. However, the hardware is steadily improving, and new algorithms for quantum computers are continually being developed. Indeed, much of the research working on quantum heuristics for combinatorial optimization is tested with classical simulations that only allow for a small number of qubits, typically around 20 qubits. Now, with larger qubit counts and devices with less noise, researchers will be able to start benchmarking these quantum heuristics at large problem sizes on quantum hardware

4.2 Proposed Workflow

- **Classical Division :** We will k-mean clustering or hierarchical clustering to divide the data set into groups.For n=100, the feasible size of the group = 20 as the IBM NISQ device can run this instance optimally.

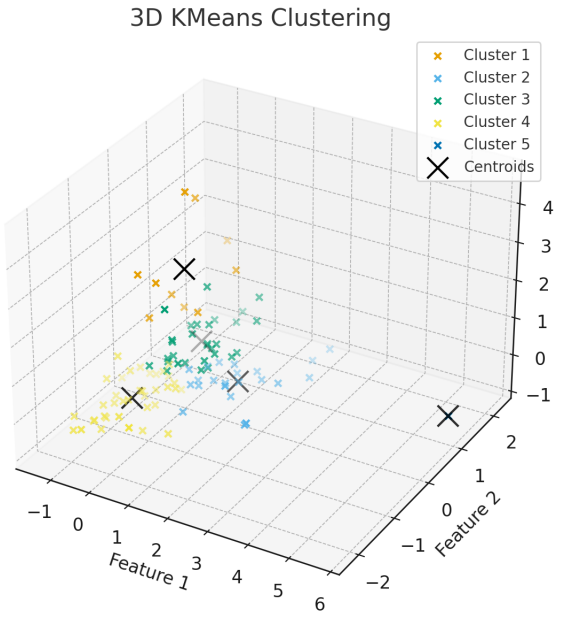

Figure 3: Group Division

- **Quantum State Manipulation by NISQ :** Then we will start with a warm solution built by a PCA matrix.Then we will apply our Hardware-Aware algorithm to find the optimal minima.

- **Quantum/Classical Post Processing :**Then we will merge our solution. Merging idea can be classical or quantum. We can use 5 qubits to encode the 5 group information to manipulate state probability.

5 Quantum Inspired solving : QIS3

QIS3 benchmarks hyperlinked

- We can implement this paper to benchmark our novel datasets over all proposed methods to compare which suits better reducing discrepancies.