0.1 问题一

0.1.1 问题背景

一般而言，QAOA 算法常被用于解决 QUBO（二次二值无约束优化）问题。然而在部分现实应用中，PUBO（多次二值无约束优化）问题也有对应求解需求。PUBO 可以直接利用 QAOA 求解，也可以利用替换变量的方式变为更多变量的 QUBO 问题。QAOA 算法在真实量子芯片运行时，一般会遇到拓扑结构问题，需要进行线路编译优化。其中，双门深度可以被当作一个比较重要的指标。

0.1.2 问题描述

给定一个 PUBO 问题：现在需要在 X 和 Y 两类材料中各挑选出 m 和 n 种材料组成某个复合材料，其中 X 类材料有 M 种，分别记作 $x_1, x_2, \cdots, x_M$，Y 类材料有 N 种，分别记作 $y_1, y_2, \cdots, y_N$。

我们使用材料的两种属性得分作为评估复合材料的依据，属性得分来自于不同 XY 材料组合得分之和。对于任意一组 (x_i, y_j) 组合，属性一得分为 α_{ij}，属性二得分为 β_{ij}，总得分则为所有 $m \times n$ 种组合对应的得分之和。目标要求最终复合材料的属性一得分尽可能接近 A，属性二得分尽可能接近 B。

例如：在 X 中选择了 $x_1, x_3, x_5 (m = 3)$，在 Y 中选择了 $y_0, y_2 (n = 2)$，则属性一得分为 $\alpha_{10} + \alpha_{12} + \alpha_{30} + \alpha_{32} + \alpha_{50} + \alpha_{52}$，属性二得分为 $\beta_{10} + \beta_{12} + \beta_{30} + \beta_{32} + \beta_{50} + \beta_{52}$。

0.1.3 问题 1

使用合适的编码手段，将问题转化为二值优化问题（不限多项式次数），并设计对应的 QAOA 算法，给出哈密顿量。

0.1.4 问题 2

在第一问基础上，以 $M=5, N=5, m=3, n=2$ 为例，在 72 量子比特、网格型的拓扑结构上实现 $p=1$ 的 QAOA 线路，包括 $Cost$ 层和 $Mixer$ 层，以 $U3+CNOT+BARRIER+MEASURE$ 作为基本门对线路进行合理排布，使得以独立 $CNOT$ 层数作为评分，$CNOT$ 层数尽可能少。选手可以随机生成 $\alpha_{ij}, \beta_{ij}, \gamma, \beta \neq 0$。

0.2 问题一解题思路

0.2.1 编码方式

我们可以为每种材料引入一个二进制变量，其中 X 类材料用 x_i 表示，Y 类材料用 y_j 表示。如果选择了某种材料，则对应的变量为 1，否则为 0。因此，对于 M 种 X 材料和 N 种 Y 材料，我们有 M+N 个二进制变量。

0.2.2 目标函数

定义目标函数为：

$$\text{Cost} = \left(\sum_{i=1}^{M} \sum_{j=1}^{N} x_i y_j \alpha_{ij} - A \right)^2 + \left(\sum_{i=1}^{M} \sum_{j=1}^{N} x_i y_j \beta_{ij} - B \right)^2 \tag{1}$$

这里，α_{ij} 和 β_{ij} 分别是材料组合 (x_i, y_j) 在属性一和属性二的得分。

0.2.3 限制条件

需要加入的限制条件是确保恰好选择 m 种 X 类材料和 n 种 Y 类材料。这可以通过以下条件实现：

$$\sum_{i=1}^{M} x_i = m, \quad \sum_{j=1}^{N} y_j = n \tag{2}$$

这些条件确保了选择的材料数量。

0.2.4 限制条件的哈密顿量

为确保恰好选择 m 种 X 材料和 n 种 Y 材料，我们引入额外的罚项：

$$H_{\text{Penalty}} = \lambda_1(\sum_{i=1}^{M} x_i - m)^2 + \lambda_2(\sum_{j=1}^{N} y_j - n)^2 \tag{3}$$

这里，λ_1 和 λ_2 是足够大的常数，用以确保在优化过程中严格遵守选择数量的限制。

0.2.5 总哈密顿量

将目标函数和罚项组合，形成最终的哈密顿量：

$$H_{\text{total}} = H_{\text{Cost}} + H_{\text{Penalty}} \tag{4}$$

这个哈密顿量既考虑了目标函数最小化，也考虑了限制条件。

0.2.6 哈密顿量构造

基于上述目标函数和限制条件，我们构造 QAOA 所需的哈密顿量。哈密顿量中每一项对应于二值变量的多项式，涵盖了目标函数和限制条件的罚函数。通过适当的线路设计，我们可以将这些项转化为量子逻辑门，实现 QAOA 算法。

0.2.7 QAOA 线路设计

1. **初态准备：**选择适合的初态，通常是所有二进制变量处于叠加态，使用 Hadamard 门将所有 qubit 置于叠加态。

2. **Cost 层设计：**设计 Cost 层来实现目标函数的相位旋转。

3. **Mixer 层设计：**Mixer 层用于在二进制变量之间引入叠加，使用 Pauli-X 门。

0.3 问题二解题思路

0.3.1 哈密顿量的构造

根据公式 (1)，*Cost* 层的哈密顿量设计如下。需要注意的是 y_j 使用的量子比特位为 $aM+j$，而不是 $M+j$，这是因为在 72 为量子芯片的网格型拓扑结构上，量子比特每 6 个成一行，为了减少 SWAP 操作，这里选择使用 $aM+j$ 的量子比特代表 y_j。我们可以使用如下函数构造 *Cost* 层的哈密顿量，它实际上完成的工作就是将公式 (1) 用量子逻辑门的形式表现出来。

```
def create_hamiltonian(M, aM, N, aN, m, n, A, B, alpha
    , beta):
    """
    :param M: x类材料种类数
    :param aM: M + 6 - (M mod 6)
    :param N: y类材料种类数
    :param aN: N + 6 - (N mod 6)
    :param m: 选中的x类材料数量
    :param n: 选中的y类材料数量
    :param A: 需要达到的目标属性一得分
    :param B: 需要达到的目标属性二得分
    :param alpha: 属性一得分矩阵
    :param beta: 属性二得分矩阵
    :return: 构造的哈密顿量
    """
    return hamiltonian
```

0.3.2 哈密顿量的重构

在 72 量子比特的网格型拓扑结构上，我们需要将哈密顿量重构为适合的顺序，以尽量减少 $CNOT$ 门的耦合与解耦合，便在量子芯片上运行。这里我们使用如下函数来重构哈密顿量，通过以下三个重排可以使得最后得到的层数最少。

```
def sort_hamiltonian(hamiltonian):

    hamiltonian.sort(key=lambda x: (min(x[0].keys())) if
        len(x[0].keys()) > 0 else -1)

    hamiltonian.sort(
        key=lambda x: np.mean((np.array(list(x[0].keys()
            )) - np.mean(np.array(list(x[0].keys()))))
            **4)
        if len(x[0].keys()) > 0 else -1)

    hamiltonian.sort(
        key=lambda x: np.mean(np.array(list(x[0].keys())
            ))
        if len(x[0].keys()) > 0 else -1)
    return hamiltonian
```

根据调试结果来看，这样的重排可以使得最后得到的层数最少。将排序函数按顺序标记为 1，2，3。以下是他们分别起作用后得到的量子线路深度：

1	2	3	depth
0	0	0	12661
0	0	1	11651
0	1	0	7748
0	1	1	1727
1	0	0	11535
1	0	1	8240
1	1	0	4264
1	1	1	1616

由此得出结论，所列出的排序函数按照顺序进行，可以得到最少的层数。

0.3.3 根据哈密顿量初步得到线路

通过上述函数，我们可以得到 $Cost$ 层的哈密顿量，并通过重排函数得到适合的顺序。接下来，我们可以将哈密顿量转化为量子逻辑门，得到 $Cost$ 层的线路。

这里对代码进行调试，截取其中一段。

```
('CNOT', [1, 0], None, None),
('BARRIER', [8, 6, 1, 0, 11, 9, 3, 2, 4, 10, 5, 7], None, None),
('U3', 0, None, (0, 0.169867582406668, 0.169867582406668)),
('BARRIER', [8, 6, 1, 0, 11, 9, 3, 2, 4, 10, 5, 7], None, None),
('CNOT', [1, 0], None, None),
('CNOT', [1, 0], None, None),
('BARRIER', [8, 6, 1, 0, 11, 9, 3, 2, 4, 10, 5, 7], None, None),
('U3', 0, None, (0, -1.34361872431639, -1.34361872431639)),
('BARRIER', [8, 6, 1, 0, 11, 9, 3, 2, 4, 10, 5, 7], None, None),
('CNOT', [1, 0], None, None),
('CNOT', [1, 0], None, None),
('BARRIER', [8, 6, 1, 0, 11, 9, 3, 2, 4, 10, 5, 7], None, None),
('U3', 0, None, (0, -1.34361872431639, -1.34361872431639)),
('BARRIER', [8, 6, 1, 0, 11, 9, 3, 2, 4, 10, 5, 7], None, None),
('CNOT', [1, 0], None, None),
```

图 1: 初始 Cost 层线路

观察容易发现，线路中存在大量反复的耦合与解耦合，这会导致线路深度过大，不利于在量子芯片上运行。因此，我们很容易得出对线路进行优化的初步方案—相同操作只进行一次耦合与解耦合。

0.3.4 线路优化

使用滑动窗口进行判断，同时需要注意首位没有前一个操作，以及末位没有后一个操作的问题：

1. 如果当前位置将要进行的操作与前一个操作不相同，且与后一个操作也不相同，则可以完整地进行 $CNOT$ 门的顺序耦合与反序解耦。

2. 如果当前位置将要进行的操作与前一个操作不相同，但与后一个操作相同，则当前操作可以只进行耦合，再使用 U3 门进行相位旋转，而不进行解耦。

3. 如果当前位置将要进行的操作与前一个操作相同，但与后一个操作不同，则当前操作可以直接使用 U3 门进行相位旋转，再进行解耦而不需要先进行耦合。

4. 如果当前位置将要进行的操作与前一个操作相同，且与后一个操作也相同，则可以只进行 U3 门的偏转，而既不需要耦合，也不需要解耦合。

函数定义如下：

```
def getCircuit(qubits, Hamiltonian, cbate, cgamma):
    """

    :param qubits: 量子比特
    :param Hamiltonian: 哈密顿量
    :param cbate: 待优化beta
    :param cgamma: 待优化gamma
    :return: 量子程序
    """
```

这样大大降低了线路的深度，提高了线路的运行效率。

0.3.5 线路拓扑结构匹配

在 72 量子比特的网格型拓扑结构上，我们需要将线路的拓扑结构与量子芯片的拓扑结构匹配，以适应量子芯片的运行。这里我们使用 *pyQpanda* 自带的函数 *topology_match* 来匹配线路的拓扑结构，所以需要重写配置文件 $QPandaConfig.json$ 以表明这是在 72 位量子芯片上进行操作。

但是由于这里只使用了 10 位量子比特进行计算，所以在配置文件中，将量子比特设置为 12 位，结构为 (2,6)，其中 x_i 使用 (0,1,2,3,4)；y_j 使用 (6,7,8,9,10)。具体位置如下图所示，只有相邻两个量子比特才能相沟通，权重都设置为 0.9。

0	1	2	3	4	5
6	7	8	9	10	11

最后再利用 *pyQpanda* 自带的 $transform\_to\_base\_qgate$ 函数，将线路转化为只包括 $U3 + CNOT + BARRIER + MEASURE$ 的基本门的量子程序即可。

0.4 列表格式输出

利用 *pyQpanda* 的 $convert\_qprog\_to\_originir$ 函数将量子程序转化为 $originIR$ 格式的长字符串，再将该字符串进行格式化输出到 *output.txt* 文件中，即可得到最终的输出结果。最后调用给出的获取深度的函数进行评分，运行结果为：

depth	1616