Universidad Internacional de la Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster en Computación Cuántica

Algoritmos cuánticos

Actividad 1 - Modelización y resolución con Dwave

Actividad de la asignatura

presentada por: Miguel Aliende García

Profesor: Mariano Caruso

Fecha: 13 de Abril de 2024

Índice de contenidos

1. Introducción **1**

2. Enunciado del problema **2**

3. Formulación del problema como QUBO **3**

4. Ejecución en QPU de D-Wave **6**

4.1. Quantum annealing 6

4.2. Ejecución en sistemas de D-Wave 6

A. Apéndices **8**

A.1. Algoritmo para pasar de grafo a QUBO 8

A.2. Calcular resultados a partir de Q 9

1. Introducción

En este ejercicio trataremos de resolver el conocido problema TSP (*Traveller salesman problem*) mediante un QPU (*Qunatum processor unit*) de (*D-Wave*).

Los QPU de *D-Wave* utilizan la técnica de *annealing* cuántico, lo que es útil para optimizar funciones expresadas como un problema tipo QUBO (*Quadratic uncosntrained binary optimization*). Por lo tanto, lo primero que haremos será expresar de manera general el problema TSP como un problema QUBO, y añadiremos las restricciones adicionales de nuestro problema em particular.

Por último utilizaremos una QPU para resolver nuestro problema QUBO.

2. Enunciado del problema

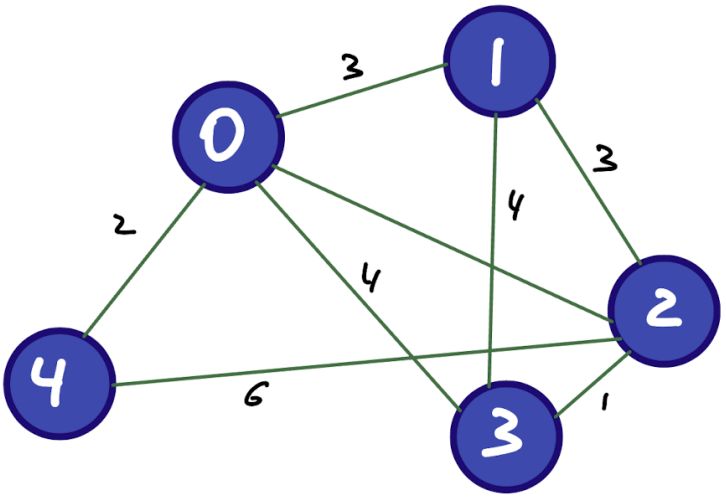

Figura 2.1: Enunciado

Cada uno de estos nodos representa una ciudad que, como podemos ver, están unidas a través de una serie de carreteras.El objetivo será encontrar la ruta de mínima distancia que cumpla las siguientes características:

- El inicio de la ruta y el final debe ser el nodo 0.
- La segunda ciudad que debemos visitar es la ciudad 2.
- Todas la ciudades deben ser visitadas.
- Solo podemos ir una vez a cada una de las ciudades.

Como hemos comentado en la introducción este grafo corresponde con el problema TSP, que se trata de un conocido problema de optimización computacional. El reto de esta clase de problema es que conforme aumentamos el número de nodos (ciudades) el número de posibles caminos crece exponencialmente y soluciones por fuerza bruta son inútiles cuando el numero de nodos crece.

3. Formulación del problema como QUBO

El primer paso es expresar nuestro problema como un problema QUBO. La razón es que esta formulación está estrechamente relacionada con el modelo Ising, un modelo matemático para estudiar propiedades ferromagnéticas a través de variables discretas que representan momentos magnéticos dipolares del *spin* atómico. Lo interesante de este modelo es que se puede mapear en procesadores basados en *annealing* cuántico mediante técnicas de computación cuántica adiabática.

En un problema QUBO tenemos una función objetivo a optimizar, que de manera general podemos describir como:

$$f(x) = \sum_{i=0}^{n} \alpha_i x_i + \sum_{i=0}^{n} \sum_{j>i} \beta_{i,j} x_i x_j \tag{3.1}$$

Donde x_i y x_j son variables binarias.

Para empezar a modelizar como QUBO, debemos entender nuestro grafo $G(N, E)$, donde los nodos N son las ciudades y las aristas E son los caminos que unen estas ciudades. Por lo tanto, sin restricciones la función debemos optimizar es:

$$\sum_{u=0}^{n-1} \sum_{j=0}^{n-1} Q_{u,j} \cdot x_{u,j} \tag{3.2}$$

Donde $x_{u,j}$ es 1 si la ciudad u es visitada en la posición j y 0 en caso contrario. Como todas las ciudades no están conectadas, vamos a plantear esta expresión para las ciudades u y v que si están conectadas, o en términos del grafo $(u, v) \in E$:

$$C = \sum_{(u,v)\in E} D_{u,v} \sum_{j=0}^{n-2} (x_{u,j} \cdot x_{v,j+1} + x_{u,n-1} \cdot x_{v,0}) \tag{3.3}$$

Donde $D_{u,v}$ representa la distancia o el coste en ir de la ciudad u a la ciudad v.

La expresión $x_{u,n-1} \cdot x_{v,0}$ la incluimos para tener en cuenta el camino de vuelta a la ciudad origen. Vamos definir ahora una función similar para aquellas ciudades que no están interconectadas, y más adelante veremos como se introducen estas restricciones en un problema QUBO, que por definición no tiene restricciones.

$$R_1 = \sum_{(u,v)\notin E} \sum_{j=0}^{n-2} (x_{u,j} \cdot x_{v,j+1} + x_{u,n-1} \cdot x_{v,0}) \tag{3.4}$$

Las siguientes restricciones que vamos a añadir son respectivamente para asegurar que cada ciudad se visita solo una única vez y que cada posición corresponde exclusivamente con una ciudad.

$$\begin{aligned} \sum_{u=0}^{n} x_{u,j} &= 1 \quad \forall j \in 0, ..., n \\ \sum_{j=0}^{n} x_{u,j} &= 1 \quad \forall u \in 0, ..., n \end{aligned} \tag{3.5}$$

Para introducir estas condiciones como restricciones vamos a usar las siguientes expresiones:

$$R_2 = \sum_{u=0}^{n} \left(\sum_{j=0}^{n} x_{u,j} - 1 \right)^2 \tag{3.6}$$

$$R_3 = \sum_{j=0}^{n} \left(\sum_{u=0}^{n} x_{u,j} - 1 \right)^2 \tag{3.7}$$

Para aplicar las restricciones, lo que haremos será introducir estas expresiones junto a 3.4 en la función objetivo multiplicadas por un valor λ arbitrariamente grande, al que se le conoce como coeficiente de Lagrange. De esta manera, este coeficiente penalizará las opciones de las restricciones, por lo que dificultará mucho que estas sean el camino óptimo.

Por último introduciremos las restricciones propias de nuestro problema concreto, que la primera ciudad sea la ciudad 0 y que la segunda en ser visitada sea la ciudad 2:

$$R_4 = x_{0,0} \tag{3.8}$$

$$R_5 = x_{2,2} \tag{3.9}$$

Por lo tanto la función objetivo que debemos minimizar es:

$$H = C + \lambda(R_1 + R_2 + R_3 - R_4 - R_5) \tag{3.10}$$

Es fácil apreciar que R_4 y R_5 son negativas para que así el coeficiente de Lagrange no las penalice, si no que, beneficie estos caminos.

4. Ejecución en QPU de D-Wave

Una vez hemos formulado nuestro problema como QUBO se lo pasamos a una QPU de D-Wave para que lo resuelva mediante la técnica de *quantum annealing*.

4.1. Quantum annealing

Las QPUs de quantum annealing están diseñadas específicamente para resolver problemas de optimización combinatoria, como nuestro TSP.

Estos sistemas simulan el proceso de enfriamiento de un material para encontrar su estado de mínima energía. Para hacer estas simulaciones, las QPUs utilizan qubits superconductores que pueden estar en un estado de superposición cuántica. Las interconexiones entre estos qubits se consiguen mediante acoplamiento electromagnético.

El proceso comienza inicializando los qubits en un estado aleatorio de alta energía, y la función objetivo a minimizar se introduce al sistema jugando con el acoplamiento entre los qubits. En este punto comienza el proceso de *annealing*, dejando que la temperatura del sistema se reduzca gradualmente y los qubits se ajusten a estados de menor energía. Cuando el sistema converge a un estado de mínima energía representa una solución óptima al problema.

4.2. Ejecución en sistemas de D-Wave

Para resolver el problema vamos a utilizar la librería *networkx* de D-Wave `https://github.com/dwavesystems/dwave-networkx/blob/0.8.14/dwave_networkx/algorithms/tsp.py#L29`, que nos facilitará introducir la función objetivo y restricciones de 3.

Empezamos creando la matriz de coste a partir de nuestro grafo con A.1 y posteriormente ejecutaremos en una QPU e interpretaremos el resultado mediante el código A.2.

El resultado de la ejecución es el siguiente:

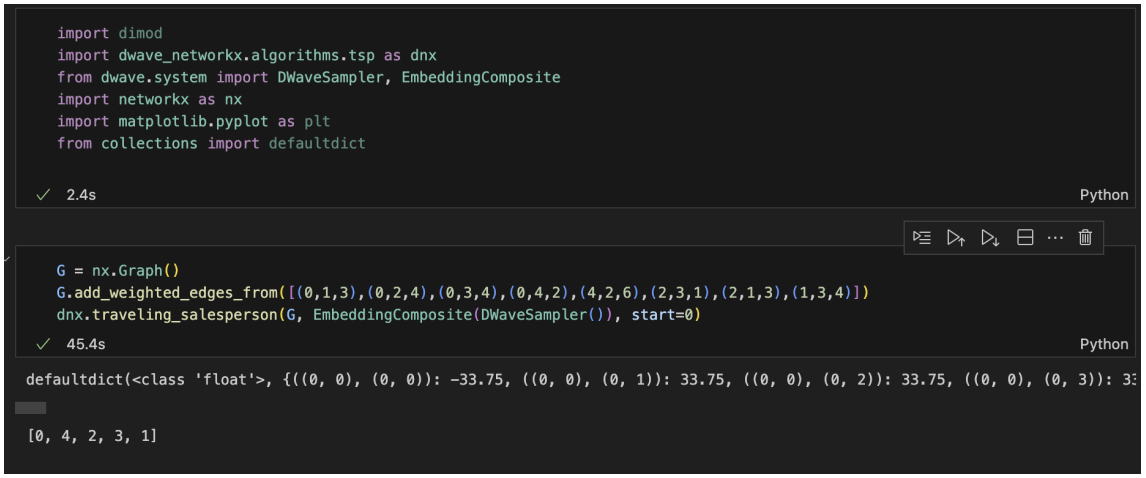

Figura 4.1: Ejecución del código

Por lo tanto, en forma de grafo, el camino óptimo es el siguiente:

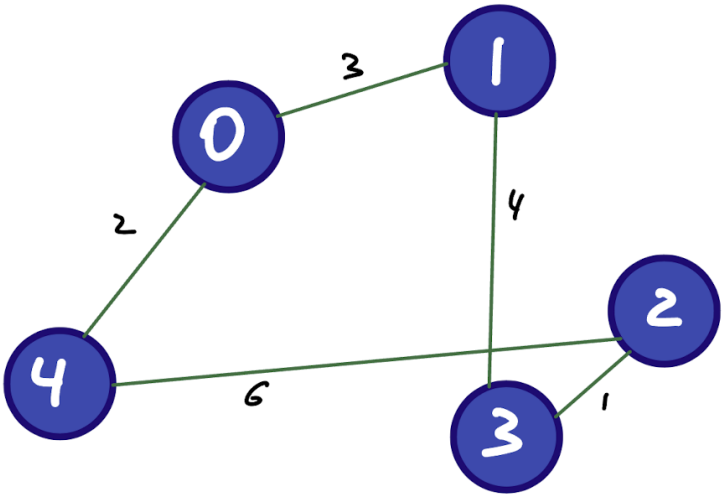

Figura 4.2: Solución por el camino 0-4-2-3-1

A. Apéndices

A.1. Algoritmo para pasar de grafo a QUBO

```
def traveling_salesperson_qubo(G, lagrange=None, weight='weight',
    missing_edge_weight=None):

    N = G.number_of_nodes()

    if lagrange is None:
        if G.number_of_edges()>0:
            lagrange = G.size(weight=weight)*G.number_of_nodes()/G.
    number_of_edges()
        else:
            lagrange = 2

    Q = defaultdict(float)

    # Constraint that each row has exactly one 1
    for node in G:
        for pos_1 in range(N):
            Q[((node, pos_1), (node, pos_1))] -= lagrange
            for pos_2 in range(pos_1+1, N):
                Q[((node, pos_1), (node, pos_2))] += 2.0*lagrange

    # Constraint that each col has exactly one 1
    for pos in range(N):
        for node_1 in G:
            Q[((node_1, pos), (node_1, pos))] -= lagrange
            for node_2 in set(G)-{node_1}:
                Q[((node_1, pos), (node_2, pos))] += lagrange

    print(Q)

    # Objective that minimizes distance
    for u, v in itertools.combinations(G.nodes, 2):
        for pos in range(N):
            nextpos = (pos + 1) % N
```

```
33
34              # going from u -> v
35              try:
36                  value = G[u][v][weight]
37              except KeyError:
38                  value = missing_edge_weight
39
40              Q[((u, pos), (v, nextpos))] += value
41
42              # going from v -> u
43              try:
44                  value = G[v][u][weight]
45              except KeyError:
46                  value = missing_edge_weight
47
48              Q[((v, pos), (u, nextpos))] += value
49
50      return Q
```

Listing A.1: Algoritmo para pasar de grafo a QUBO

A.2. Calcular resultados a partir de Q

```
1  def traveling_salesperson(G, sampler=None, lagrange=None, weight='weight',
2                            start=None, **sampler_args):
3
4      Q = traveling_salesperson_qubo(G, lagrange, weight)
5
6      # use the sampler to find low energy states
7      response = sampler.sample_qubo(Q, **sampler_args)
8
9      sample = response.first.sample
10
11     route = [None]*len(G)
12     for (city, time), val in sample.items():
13         if val:
14             route[time] = city
15
16     if start is not None and route[0] != start:
```

```
17          # rotate to put the start in front
18          idx = route.index(start)
19          route = route[idx:] + route[:idx]
20
21      return route
```

Listing A.2: Algoritmo para pasar de grafo a QUBO