1 Introducción

El problema del agente viajero consiste en encontrar la ruta cerrada de menor costo, donde cada ciudad es visitada una vez antes de volver al punto de partida. A continuación se presenta la formulación del problema mediante programación lineal.

$$\text{Variables de desición: } x_{ij} = \begin{cases} 1 & \text{la ruta va de la ciudad i a la j} \\ 0 & \text{EOC} \end{cases}$$

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, ..., n \tag{1}$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{2}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n \tag{3}$$

$$1 \leq u_i \leq n - 1 \qquad 2 \leq i \leq n \tag{4}$$

$$u_i \in \mathbb{Z} \qquad i = 2, ..., n$$
$$x_{ij} \in \{0, 1\} \qquad i, j = 1, ..., n$$

En este proyecto $c_{ij} = c_{ji}$ y representa la distancia euclidiana entre el nodo i y el j. El primer conjunto de igualdades (1) asegura que la ruta pase por todos los nodos, el segundo (2) garantiza que desde cada ciudad haya un viaje a otra ciudad. Los últimos dos conjuntos de desigualdades (3) y (4) evitan que se generen rutas disjuntas.

2 Heurístico del vecino más cercano

Decidimos utilizar el heuristíco del vecino más cercano para encontrar la ruta óptima. Ya que, debido a la complejidad computacional del problema y el tamaño de los datos resulta muy ineficiente solucionar el problema con el algoritmo exacto (tendría una complejidad de $n!$). El algoritmo del vecino más cercano funciona de la siguiente manera, desde un nodo inicial arbitrario x_i se calculan las distancias a todos los nodos que no se han visitado y se selecciona al más cercano entre estos. Repetimos lo anterior hasta haber visitado todos los nodos. Para evitar una ruta poco óptima debido a un nodo inicial no conveniente, corremos el algoritmo una vez para cada nodo x_i como nodo inicial. A continuación, el pseudocódigo recursivo.

Algorithm 1: nearest-neighbor

Input: coordenada
Output: [ruta-optima, distancia-optima]

```
1  matriz-distancias ← matriz de distancias entre cada punto;
2  N ← cantidad de coordenadas;
3  ruta-optima ← [ ];
4  distancia-optima ← ∞ ;
5  for (i = 0; i < N; i++) do
6  |  distancia ← 0;
7  |  no-visitado = ones[N];
8  |  no-visitado[i] = 0;
9  |  if Σ no-visitado > 0 then
10 |  |  [ruta, distancia] = nearest-neighbor2(i, no-visitado, ruta, distancia,
   |  |   matriz-distancias);
11 |  end
12 |  distancia = distancia + distancia del último nodo al primero;
13 |  if distancia < distancia-optima then
14 |  |  distancia-optima = distancia;
15 |  |  ruta-optima = ruta;
16 |  end
17 end
18 return [ruta-optima, distancia-optima]
```

Algorithm 2: nearest-neighbor2

Input: [i, no-visitado, ruta, distancia, matriz-distancias]
Output: [ruta, distancia]

```
1  nn-distancia ← minimo de distancias entre nodo_i y nodos no visitados;
2  nn-indice ← indice de nn-distancia;
3  not-visited[nn-indice] = 0;
4  if Σ no-visitado > 0 then
5  |  [ruta, distancia] = nearest-neighbor2(nn-distancia, no-visitado, ruta, distancia,
   |   matriz-distancias);
6  else
7  |  ruta = [ruta, nn-indice]
8  end
9  ruta = [ruta, i] distancia = distancia + nn-distancia;
10 return [ruta, distancia]
```

3 Prueba y resultados

Se probó el algoritmo en una base de datos con coordenadas en Qatar con 194 puntos, enumerados del 0 al 193.

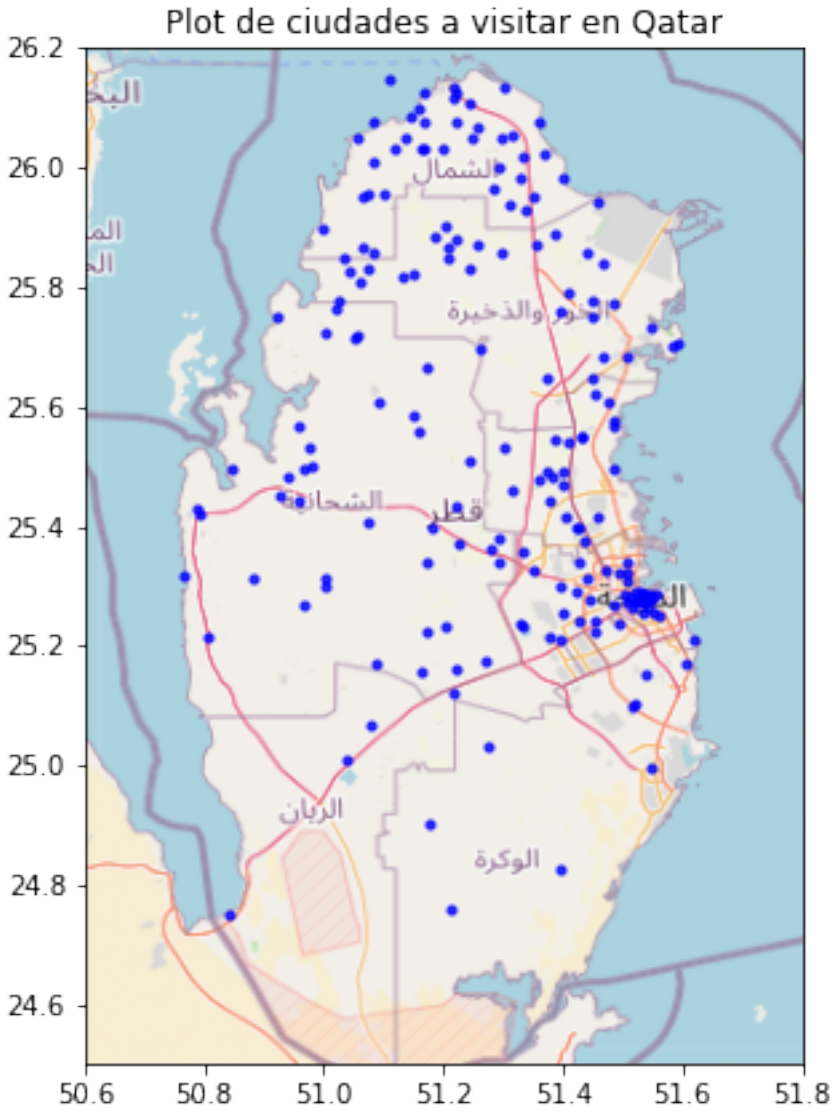

Figure 1: Gráfica de las 194 ubicaciones en Qatar

Al correr el algoritmo obtuvimos los siguientes resultados.
Ruta optima: [19, 6, 3, 1, 2, 4, 8, 9, 11, 14, 18, 159, 193, 175, 181, 182, 185, 186, 189, 191, 188, 190, 187, 192, 184, 170, 165, 161, 157, 158, 164, 167, 166, 169, 179, 177, 176, 180, 183, 174, 172, 173, 178, 171, 168, 163, 162, 160, 155, 129, 97, 85, 84, 64, 62, 35, 61,58, 22, 24, 70, 75, 86, 79, 81, 88, 89, 93, 98, 100, 103, 110, 118, 121, 117, 99, 83, 44, 56, 59, 68, 73, 71, 74, 77, 90, 102, 101, 108, 112, 113, 125, 124, 126, 131, 133, 136, 139, 144, 148, 145, 141, 137, 138, 140, 156, 153, 143, 149, 152, 151, 146, 150, 154, 142, 147, 135, 130, 128, 134, 132, 123, 122, 127, 119, 120, 116, 115, 114, 111, 109, 107, 106, 105, 104, 96, 92, 95, 94, 91, 87, 82, 80, 78, 76, 69, 63, 67, 72, 65, 66, 60, 53, 49, 54, 48, 43, 41, 34, 29, 31, 30, 33, 39, 42, 37, 40, 45, 47, 51, 52, 55, 57, 50, 46, 38, 36, 26, 21, 28, 27, 32, 17, 20, 23, 25, 16, 10, 13, 12, 15, 7, 5, 0, 19]
Cuya distancia total fue de 11621.96.
El algoritmo tardó 65.509 segundos.
Con un tiempo por iteración (posible solución) de 0.3377 segundos.
Se anexa al final del documento el código utilizado en Python.

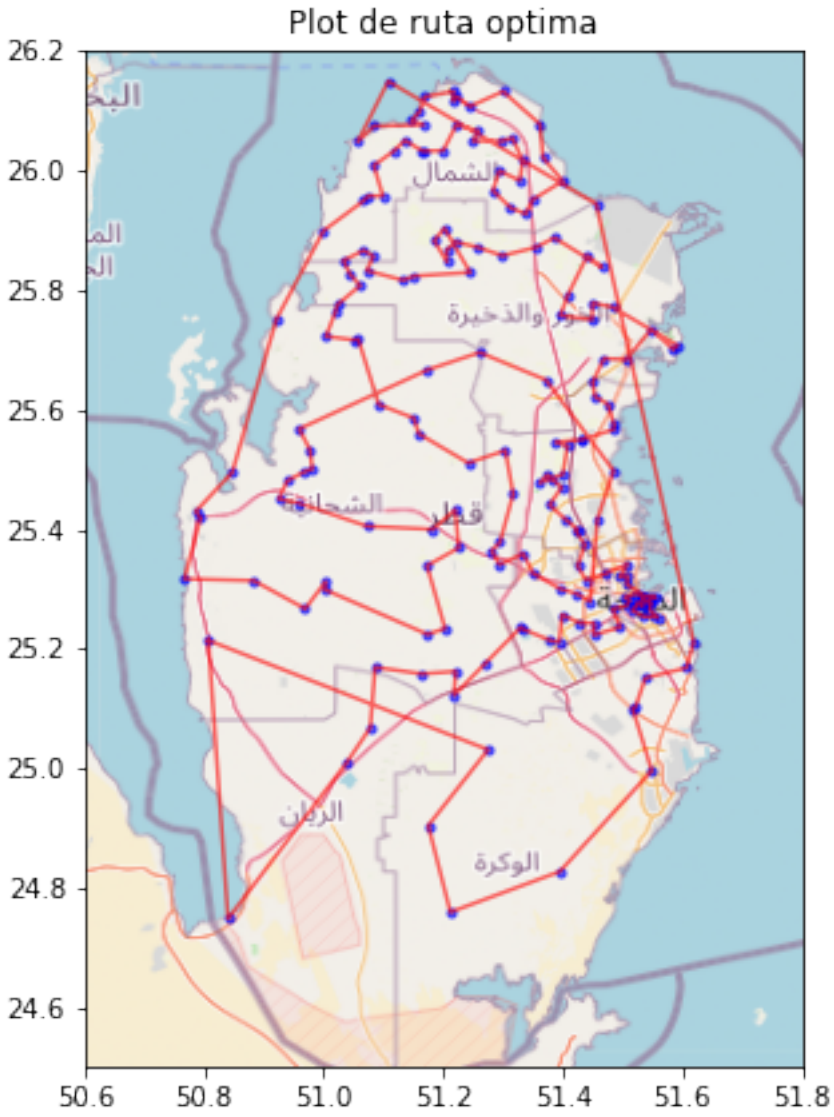

Figure 2: Ruta optima