Documentación técnica

Andrés Urbano Guillermo Gerardo

3 de Diciembre del 2021

1. Planteamiento del problema a resolver

Se desea encontrar las rutas para los turísticas que visiten la Ciudad de México de tal modo que visiten los mercados tradicionales de sus alcaldías y pueden recorrer la mayor parte de las calles sin tener que pasar por la misma calle, de este modo, pueden disfrutar de las calles y mercados de la Ciudad de México aprovechando más su tiempo viendo las artesanías y degustando la comida mexicana que se puede encontrar en los diferentes mercados. Por otra parte, también se desea resolver el problema de los distribuidores que entregan la mercancía a los diferentes mercados de cada delegación de tal modo que pueden ir sin tener que pasar por el mismo mercado dos veces para llegar al siguiente, buscando la ruta óptima y recomerdasela, de es este modo, podemos minimizar el tiempo y recursos en transporte.

2. Forma de modelarlo matemáticamente y justificación de suposiciones

Para modelar nuestros problemas matemáticamente primeramente vamos pensar primero en que cada mercado de la ciudad de México se puede representar como un nodo de una gráfica, y la conexión entre cada mercado será representada por una arista con un peso que indique la distancia que se requiere en auto para llegar de un mercado a otro.

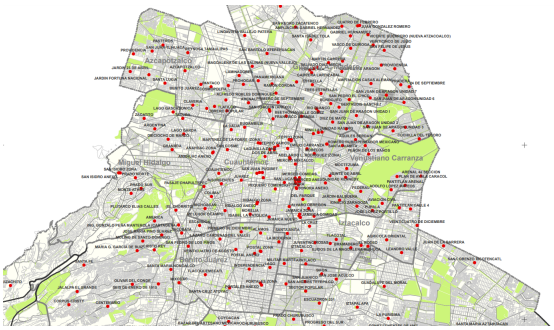

Figura 1: Puntos de diferentes mercados CDMX

Partiremos con dos conceptos importantes para encontrar la solución de estos problemas , primero partiremos en definir sobre los **grafos eulerianos y hamiltoniamos**

LLamaremos camino euleriano a un camino que contiene a todas las aristas del grafo, apareciendo cada una exactamente una vez. Un ciclo euleriano es un camino euleriano que comienza y acaba en el mismo veŕtice.

Definicio. Un grafo que admite un ciclo euleriano diremos que es un grafo euleriano.

Un camino hamiltoniano es un camino que recorre todos los vertices de un grafo sin pasar dos veces por el mismo vertice. Si el camnino es cerrado se dice un ciclo hamiltoniano.

Definicio. Un grafo G se dice hamiltoniamo si tiene un ciclo hamiltoniano.

Enseguida enunciaremos algunas proposiciones auxiliares que será de ayuda para nuestro problema:

Proposición. Si G es un gráfica para la cual existe un paseo que pasa por todas las aristas, entonces debe existir un camino que pasa por todas las aristas.

Proposición. Una trayectoria que pasa por n vértices pasa exactamente por n - 1 aristas.

Partiremos con estas preposiciones podemos ver que nuestro problema acerca de los repartidores de mercancía y turistas se enfoca en encontrar un grafo que sea hamiltoniano de tal modo que la construcción del grafo cada arista tenga el peso mínimo. Veremos más adelante que parecer ser una solución sencilla sin ningún impedimento, pero conforme analizamos más datos veremos que este problema tiene algo especial.

3. Enunciado del problema algorítmicos formal general a resolver

El problema a enunciar consiste en encontrar la ruta optima que pase por todos mercados de cada delegación del tal modo que pase sólo una vez. A primera instancia podemos pensar en resolver este problema utilizando fuerza bruta, buscando todas las rutas posibles, pero esto podría darnos una complejidad O(n!) por buscar todas permutaciones posibles, enseguida pensaremos en otras técnicas para resolver nuestro problema.

4. Propuesta de solución mediante algoritmos combinatorios

Como mencionamos anteriormente nuestro problemas puede ser resuelto sencillamente con fuerza bruta haciendo una búsqueda exhaustiva, pero dado que tenemos varios mercados la complejidad en tiempo aumenta, esto puede verse utilizando la técnica de backtraing, esto nos encontraría la mejor solución a nuestro problema ya que conoceriamos todos los posibles caminos de llegar de un mercado a otro sin tener que pasar por el, y la forma de pseudocodigo seria:

```
ruta_optima_backtrack(nodo_inicial, G, n,a=[], visitados=[], dist=0):
    # Significa que ya ha revisado cada arista del grafo (n-1 aristas)
    if len(a) == n-1:
        #print('Camino: {} distancia total: {}'.format(a, dist))
        caminos.append(a.copy())
        dist_totales.append(dist)
        return dist

    candidatos = G.adj[nodo_inicial]
    for vecino, peso in candidatos.items():
        if vecino not in visitados:
            a.append((nodo_inicial, vecino, peso['weight']))
            visitados.append(nodo_inicial)
            ruta_optima_backtrack(vecino, G, n, a, visitados, dist+peso['weight'])
            # Paso hacia atras
            a.remove((nodo_inicial, vecino, peso['weight']))
            visitados.remove(nodo_inicial)
```

pero esto podria causarnos problemas cuando tenemos la entrada muy grande, más adelante hableremos sobre su complejidad y responderemos la pregunta de por que es una buena opción, otra opción a elegir seria utilizar un algoritmo voraz (greedy) que vaya tomando la mejor decisión para optimizar nuestra ruta, ahora este enfoque tiene un inconveniente que puede no ser la opción más óptima, ya que toma siempre la ruta con menor distancia , pero no considera el pasado y el futuro de nuestra gráfica, esto puede llevar a un camino nada optimo.

```
def ruta_optima_greedy(nodo_inicial, G):
    n_node = G.number_of_nodes() - 1
    ruta_optima = []
    nodos_visitados = []
    nodos_visitados.append(nodo_inicial)
    while(n_node != 0):
        peso_minimo = float("inf")
```

```
        nodo_siguiente = None
        for vecino, peso in G.adj[nodo_inicial].items():
            if peso['weight'] < peso_minimo and vecino not in nodos_visitados:
                peso_minimo = peso['weight']
                nodo_siguiente = vecino

        ruta_optima.append((nodo_inicial, nodo_siguiente, round(peso_minimo, 3)))
        nodos_visitados.append(nodo_siguiente)
        nodo_inicial = nodo_siguiente
        n_node -= 1
    return ruta_optima
```

Por otra parte, la complejidad será mejor cuando la entrada es demasiada grande, por lo que no descartaremos esta opción.

5. Análisis de correctitud y análisis asintótico de tiempo y espacio

Para hablar de complejidad en tiempo y espacio, hableremos un poco sobre la idea de resolver este problema utilizando backtrinkg. La tecnica backtraing se enconfocara encontrar todas las combinaciones posibles que puede haber en nuestra gráfica, es decir, buscara todos los caminos posibles, esta tecnica utilizara la recursion para hacer el movimiento hacia atras en un arbol de recursión, y este es uno de problemas que podemos ver en espacio, dado que será recursivo ocupara mucha memoria RAM dependiendo de la entrada de nuestro problema, veremos más adelante que precisamente al intentar hacer backtring en un conjunto de mercados la memoria se acaba y no se puede proseguir, por otro lado hablemos de en comlejidad de tiempo, dado que el algoritmo buscara todas combinaciones posibles tendra una complejidad $0(!n)$ en tiempo y espacio, esto puede verse cuando tenemos una entrada de 10 mercados, esto nos llevara a tener 3628800 rutas optimas, vemos como la complejidad va aumente comporme tenemos más datos, por eso mismo, este problema llegar ser conocido como un problema NP-difícil debido a que no existe un algoritmo de tiempo que lo resuelva de manera eficiente.

Ahora analicemos cuando utilizamos un algoritmo voraz para encontrar nuestra solución, en primera instacia puede verse que no será una opción por tener una complejidad en tiempo $O(n^2)$ si consideramos que nos toma primero n pasos explorar a cada uno de los vecinos de un nodo y otros n pasos para ver el siguiente nodo respectivamente. Además de eso, como mencionamos anteriormente puede no nos darnos la mejor solución, pero veremos más adelante que este tiempo polinomial es muy bueno cuando tenemos conjunto de entradas no tan grande que puede lineal si nos acercamos con una lupa a su gráfica complejidad que puede resultar bueno para entradas no tan grandes.

Otra opción a pensar sería utilizar la programación dimamica, ya que podemos guardar ciertas rutas y usar la técnica de memorización para ir recortando nuestro espacio de estado, como se hace en un arbol de fibonacci.

6. Estrategias de diseño de algoritmos o estructuras de datos utilizadas

La primera a estregia a utilizar será sera representar a nuestro conjunto de datos con una estrutura de dato Grafo, ya que necesitaremos crear una gráfica con pesos para encontrar nuestra optima y probar dos diferentes algoritmos, uno con el modelo greedy y otro con el modelo de fuerza bruta.

7. Aplicación a los datos concretos.

Como mencionamos en nuestro anteriorment, queremos fomentar el turismo en los mercados de la Ciudad de México y ayudar a los provedores a llevar la mercancía en tiempo y forma a los mercados de las diferentes delegaciones. Para necesitamos encontrar la ruta de ayude a pasar a cada uno de los mercados sin tener que volver a pasar por el mismo.

Primero decimos el criterio para conocer las distancia entre los diferente mercados de la CDMX, para eso tendremos que utilizar la API de Google para generar una matriz de distancias, en nuestro caso, dado que google pide una suscripción con tarjeta de credito para usar sus servicios, mejor decidi utilizar otra alternativa con una biblioteca llamada 'geodesic' que nos dara distancias segun su ubicación en coordenadas geografias. Esta seria una buena aproximación en análogo a la API de Google, ya que lo importante es conseguir los pesos en nuestra gráfica, nos enfocaremos hacer el modelo y el algortimo, dado que si funciona el algoritmo con cualquier distancia, tiempo o cualquier matriz que valide el peso de grafica con sus respectivas aristas, funcionará para cualquier entrada, siendo independiente del algortimo.

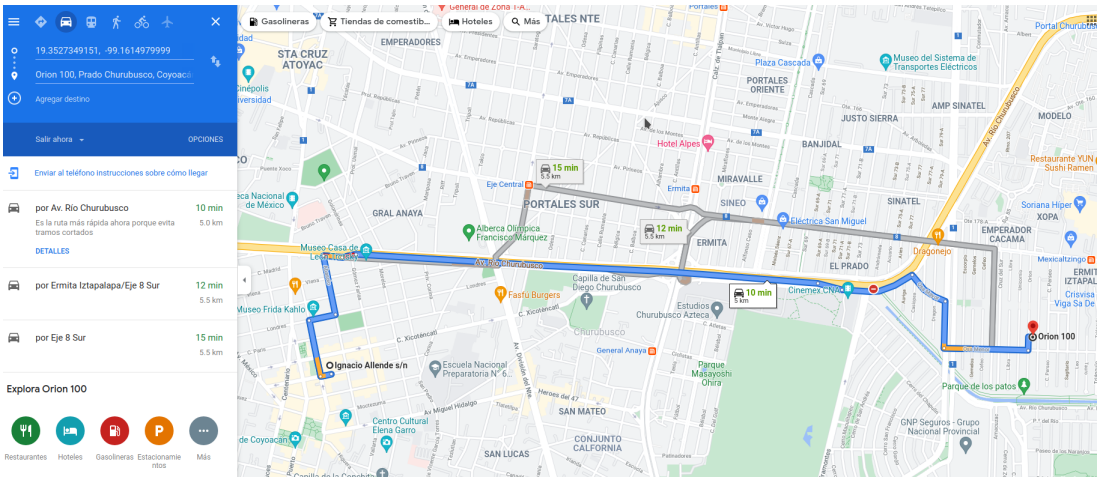

Figura 2: Distancia entre los mercados

Esto se puede ampliar usando diferentes distancias como por ejemplo no es lo mismo la distancia de ir en auto que caminando, esto varia y por lo tanto grafica puede variar.

Defineremos una función para calcular la entrada:

```python
def crear_matriz_distancias(mercados):
    """Crea una matriz de distancias"""
    matriz_distancias = np.zeros((len(mercados), len(mercados)))
    for i, mercado1 in enumerate(mercados):
        for j, mercado2 in enumerate(mercados):

            if mercado1 == mercado2:
                dist_ = float('inf')
                matriz_distancias[i][j] = dist_
                continue
            # Aprovechando la simetria
            if j > i :
                dist_ = obtener_distancia(mercado1, mercado2)
                matriz_distancias[i][j] = dist_
                matriz_distancias[j][i] = dist_

    return matriz_distancias
```

Una vez definido nuestra función que sacara la distancia entre 2 mercados, tendremos que sacar ahora la matriz de distancia de todos los mercados, esto nos llevaria un tiempo de complejidad de $O(n^2)$, utilizaremos heuristicas de optmización aprovechando la simetria de la matriz, ya que en nuestro

proble el qir de A a B y de B a A será el mismo camino. Con esto los calculos se reduciran a la mitad, calculando solamente la matriz superior de nuestra matriz, y la diagonal principal que estará llena de infinitos dado que no queremos que interfiera en nuestro analisis.

Primero haremos un caso particular en una delegación, para despues hacer un caso general para cada uno de los mercados de las diferentes delegaciones:

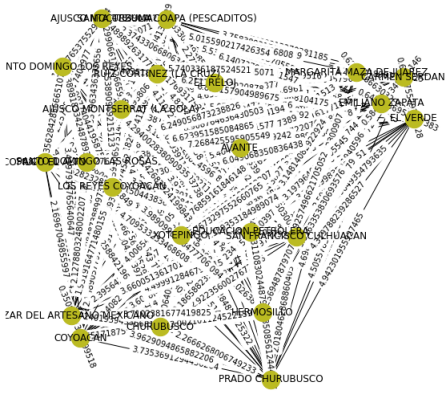

Figura 3: Mercados de la delegación de Coyoacan

Utilizaremos la idea de programación Orientada a Objetos para abstraer la información más importante para nuestro problema, en nuestro caso, crearemos una clase Mercado que contendra las coordenadas geograficas del mercado, como su respectivo nombre.

Utilizaremos un algoritmo greedy como mencionamos en el documento, ya que tiene un tiempo de complejidad polinomial $O(n^2)$ que puede verse lineal cuando tenemos un conjunto de entrada pequeños, en nuestro, nuestro conjunto de entrada corresponde a los mercados de cada delagación lo cual no pasan de 50, probaremos este algoritmo y veremos los resultados:

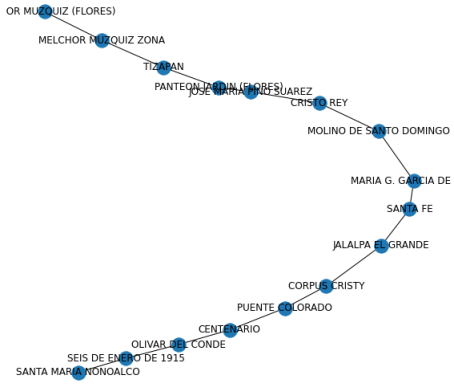

Figura 4: Camino óptimo delación Coyoacan

Si bien esta ruta podria no ser la optima, debido a que toma sólo los mejores pesos en el momento, sin considerar el pasodo o futuro como otros algortimo, pero nos da una aproximación a una ruta en tiempo relativamente rápido por el tamaño de nuestra entrada, enseguido mostraremos un algortimo con la metodologia de backtracking.

Utilizar backtracking nos pueda mejores resultado ya que obtedremos todos los posibles caminos, pero esto nos lleva a explorar por fuerta bruta todo el conjunto del espacio estado y esto puede provocar una complejidad elevada en tiempo y espacio.

Con esta paranamica podemos observar la distribución de los diferentes mercados de la Ciudad de mexico. Como hemos anteriormente deseamos conocer la ruta más optima que pase por cada uno

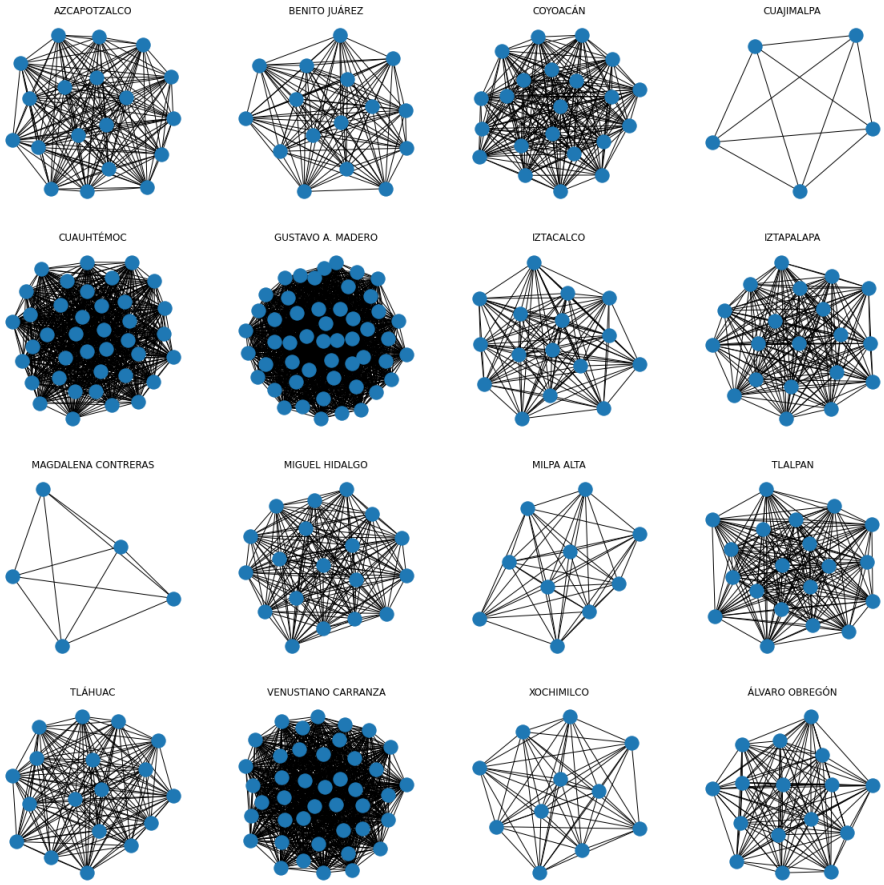

Figura 5: Mercados de todas las delegaciones

de los mercados, para eso decidimos utilizar el algoritmo creado anteriormente con la modelo greedy dado que nuestro conjunto de entrada no es tan grande por cada delegación, por lo tanto puede verse lineal si lo vemos con una lupa, por eso mismo hemos decidimos usarlos para la solución de nuestro problema.

Por último, aplicaremos nuestro algoritmo en todas las delagaciones para conocer la ruta optima, veremos que este puede ser de gran ayuda primeramente a poder explorar a todos los mercados de manera turisticas y conocer más de ellos. Por otro, los repartidores que lleva la mercancía a los diferentes mercados de todas las delegaciones puede ser de gran utilidad ya que puede ahorrar recursos en gasolina, como también en tiempo si llega más rapido a su destino.

8. Conclusiones y posible trabajo a futuro.

Para concluir vemos que este problema tambien conocido como problema del agente viajero no es tan facil como parece, ya que puede resultar encontrar la solución en un tiempo muy extension encontrando la mejor solución, como vimos si queremos encontrar la mejor solución se requiere tecnicas de programación dimica para reducir el espacio de estados, ya que si utilizamos solamente backtrack esto tardaria $O(!n)$ de tiempo, por lo que no resultaria util, decidimos utilizar el algoritmo greedy por eficacio en nuestra datos ya que los resultados que nos devuelve son rapidos, tal vez no sea la más optima, pero como mencionamos anteriormente es un problema NP-dificil que puede dartas mucho tiempo para encontrar la mejor solución.

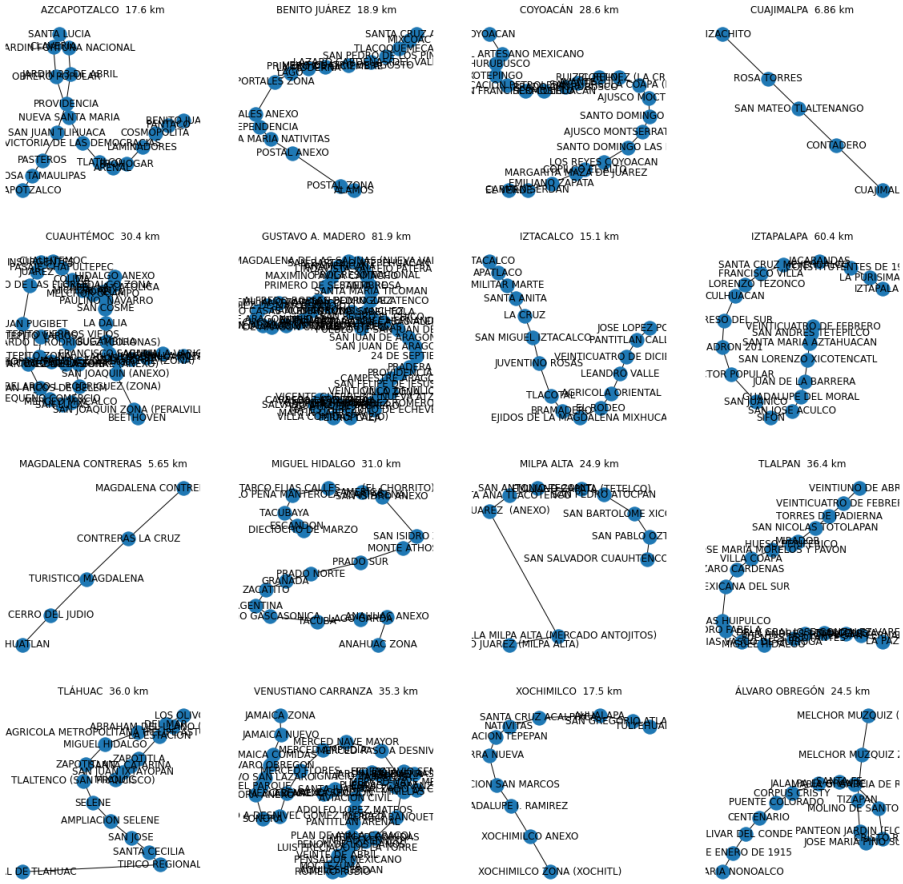

Figura 6: Caminos más óptimos de todas las delegaciones

9. Bibliografía.

Referencias

[1] Flórez, Jairo Alberto Villegas: *Una aplicación del método MTZ a la solución del problema del agente viajero.*

[2] *https://stackoverflow.com/questions/45479728/calculate-distance-between-2-points-in-google-maps-using-python.*

[3] https://youtu.be/W16E5qzzCxU. Accedido en 2 Diciembre del 2021.

[4] https://youtu.be/O9l8ultBAig. Accedido en 2 Diciembre del 2021.

[5] https://developers.google.com/optimization/routing/tsp. Accedido en 2 Diciembre del 2021.