Matematicas
Discretas

Análisis de ecobicis

Proyecto Final

Acosta Imandt Daniel
Castillo del Valle Pablo David

3/Diciembre/2021

Estructuración del proyecto

Para un entendimiento sencillo de los pasos utilizados en este problema, este documento esta dividido en documentación ejecutiva y la documentación técnica, dependiendo de lo que se trate de ver.

Por otro lado este escrito esta pensado para leerse al mismo tiempo que el siguiente Notebook que es en donde se encuentran todas las implementaciones que hicimos para el análisis

Así mismo si esta interesado en ver otro proyecto que se hizo con estos datos, en donde vimos el tipo de persona que usa este sistema de transporte, los días y horas que se frecuenta más y el gran cambio que creo el Covid para su uso, puedes ver este proyecto:
Notebook trabajos pasados

Documentación ejecutiva

Planteamiento del problema

Existen grandes problemas en la Ciudad de México, entre ellos se encuentra la contaminación con grandes cantidades de CO2 en el ambiente, los cuales en gran parte son generados por autos particulares.
Otro problema que molesta a una gran cantidad de capitalinos es el trafico que suele haber en horas pico, ya que según la plataforma "Sin Tráfico"los adultos en la Ciudad suelen perder 6 días al año atrapados en el tráfico.[1]
Por lo que es muy necesario buscar soluciones a estos problemas.Notamos que si más personas utilizan transporte publico y menos autos particulares, se lograría avanzar a una solución.
En la ciudad existe el sistema de transporte de ecobici el cual se suele encontrar en una pequeña parte de esta , por lo que lamentablemente no puede llegar a más usuarios que lo requieren, por lo que existe la pregunta de que tan conveniente seria aumentar las estaciones de ecobici a lo largo de la ciudad y también tratar de ver el tiempo promedio que se tardaría un usuario de ir de cualquier estación a la otra siguiendo la ruta dada por el algoritmo, esto con la finalidad de demostrarle a más personas que los trayectos suelen ser bastante rápidos, y que al utilizar el sistema pueden ahorrar tiempo siguiendo la ruta propuesta por nuestro algoritmo .

Obtención lo los datos

Por lo comentado anteriormente nació nuestro interés en estudiar redes de transporte de la Ciudad de México, en particular el uso de bicicletas compartidas.Así que en este proyecto decidimos utilizar los datos obtenidos por la compañía de ecobici,de sus viajes diarios (Estos datasets se pueden conseguir en la siguiente liga(https://www.ecobici.cdmx.gob.mx/es/informacion-del-servicio/open-data).Así mismo estamos interesados en conocer las localizaciones de las estaciones ecobici donde se pueden conseguir en este liga(https://datos.cdmx.gob.mx/dataset/estaciones-de-ecobici), en el cual a la hora de escribir esto nos dimos cuenta que ya no estaba disponible, por lo que adjuntamos los datos previamente descargados.
Analizaremos estos datos para tratar de describir el comportamiento de los usuarios de este sistema, para ver si es requerido poner mas estaciones en las periferias y calcular el tiempo estimado de una estación a otra.
Aquí anexamos la carpeta con todos los datos que se usaron, por si no los encuentran en los anteriores: Carpeta de datos

Análisis

A continuación vienen los pasos que se siguió y los resultados obtenidos.

1. Exploración de los datos
2. Limpieza de los datos
3. Calculamos los tiempos de cada trayecto
4. Vemos las estaciones que tienen una mayor centralidad
5. Encontramos la ruta más rápida dados los datos que tenemos
6. Decimos el tiempo aproximado de viaje y mostramos en un mapa un aproximado del camino

Primero queremos ver si hay algún cambio significativo del tiempo promedio de viaje para las distintas horas del día, vemos que solo en la noche hay una pequeña disminución de los tiempos, y en las demás horas parece ser el mismo.Pero aún así vamos a ver una pequeña implementación de nuestro algoritmo tomando las horas del día.
A continuación vemos las estaciones que tienen mayor centralidad.

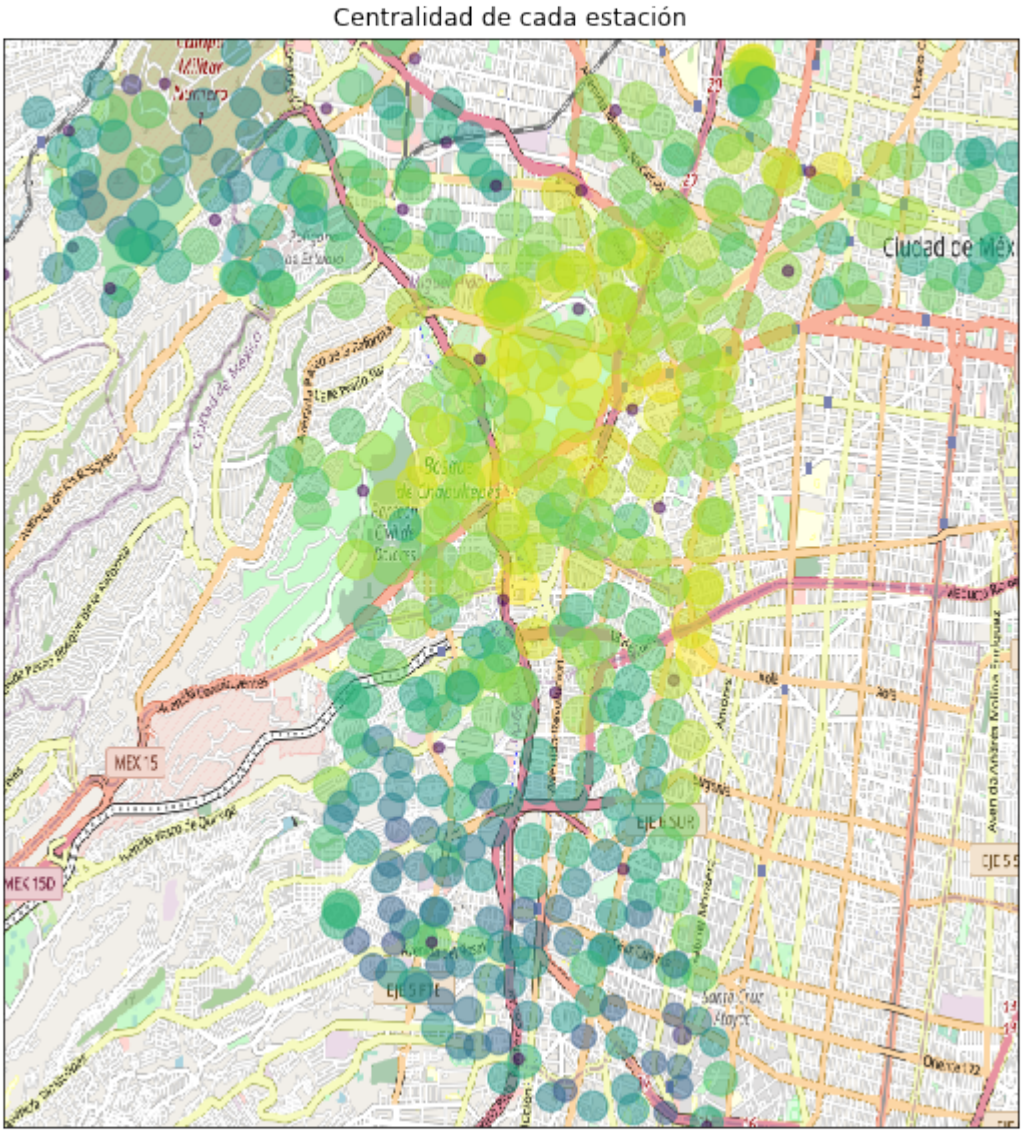

Figura 1: Importancia de cada estación

Como parece indicar la gráfica todos los nodos se ven bastante conectados unos con otros.

La intensidad de color de cada nodo dice su centralidad de eigenvector la que toma en cuenta la importancia de sus vecinos también.
El tamaño de cada nodo nos dice la centralidad de grado, en donde vemos cuantos vecinos tiene cada nodo.
Como podemos ver a partir de la imagen los nodos que tienen una mayor centralidad de eigenvector, son los que se encuentran casi en el centro, lo cual tiene sentido ya que es la sección que esta más conectada con las otras secciones geográficamente hablando.
Por otro lado viendo el tamaño de los nodos parece indicar que casi todos son del mismo tamaño , es decir todos tienen muchos vecinos, por lo que esta fuertemente conectada.

Lo que haremos ahora será agarrar alrededor del $1\,\%$ de todos los viajes y graficarlos, para darnos

una idea de como es su comportamiento.

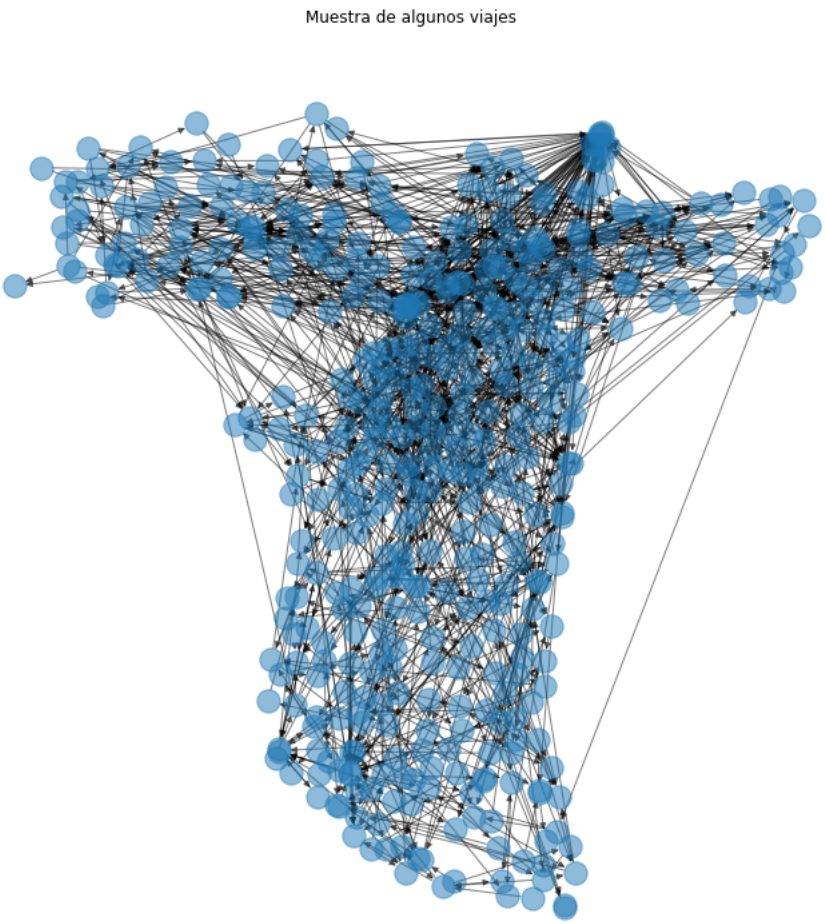

Figura 2: Muestra de algunos viajes

Por ultimo veremos algunos ejemplos de como se comporta nuestro algoritmo y los tiempos que se nos da para dos estaciones

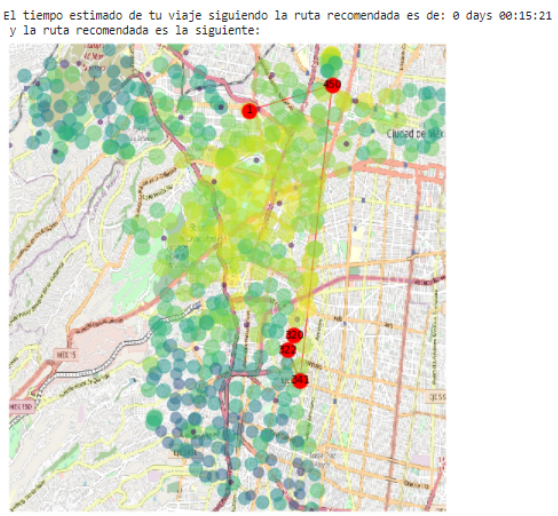

Figura 3: Tiempo estimado más la ruta recomendada de la estación 1 RIO SENA-RIO BALSAS a la 320 ENRIQUE RÉBSAMEN-LUZ SAVIÑON

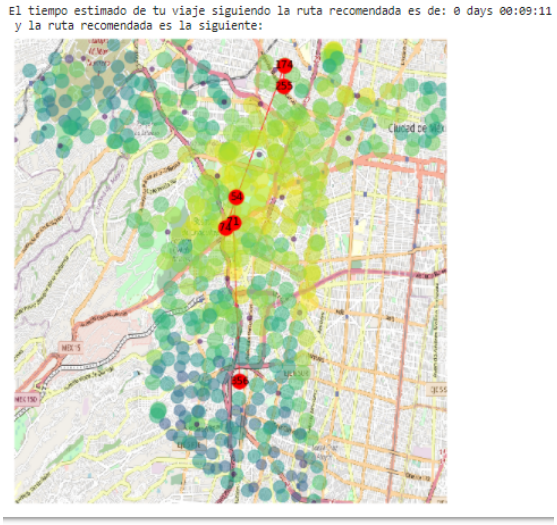

Figura 4: Tiempo estimado más la ruta recomendada de la estación 255 JOSE ROSAS MORENO-GUILLERMO PRIETO a la 356 MATIAS ROMERO -ADOLFO PRIETO

Como podemos ver los dos ejemplos nos dan buenos tiempos para trayectos que abarcan casi toda la altura de la gráfica, ya que la de la estación 1 a la 320 se tardo 15 minutos con 21 segundos,en comparación con los 31 minutos que te tardarías en coche según Google maps. Y de la 255 a la 356 fueron 9 minutos con 11 segundos comparados con los 34 minutos que dice que haces en coche según google maps, lo único que tal vez es el problema es que pareciera ser que nos hace dar vueltas innecesarias en vez de tomar el camino más directo, esto se puede deber a que en efecto es más rápido de esta forma o que pueden faltar datos para que la distancia en número de vértices sea menor. Ahora cabe destacar que al implementar el algoritmo estamos considerando el promedio del tiempo de viaje, sobre cada viaje hecho entre dos nodos, sin embargo, esto no necesarimanete es óptimo ya que, puede que la hora del día sea un factor importante en el tiempo de los recorridos. Observemos primero que esto es cierto:

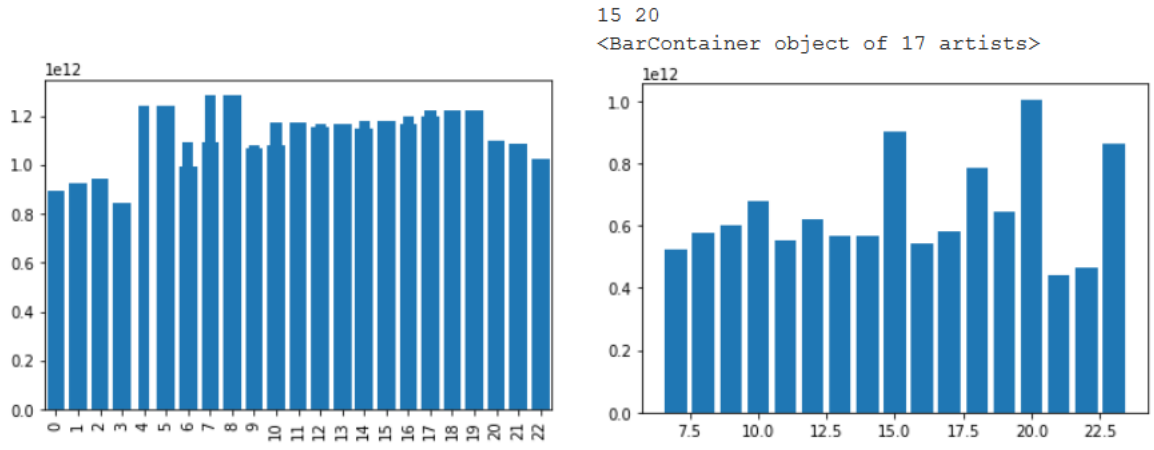

Figura 5: A la izquierda el promedio del tiempo de trayecto según la hora del día promediando todas las estaciones, a la derecha el promedio para los recorridos de la estación 15 a la 20

Como podemos notar aunque en primera instancia el tiempo promedio de la duración de los recorridos para cualquier hora del día, cuando checamos específicamente dos el recorrido entre dos estaciones, podemos notar que hay horas en el día en las que la diferencia de tiempo es notablemente baja por lo tanto la ruta óptima puede no ser la que pensamos anteriormente. Veamos un ejemplo en el que la ruta óptima cambia con respecto a la hora del día:

```
A las 9:00 horas
El tiempo estimado de tu viaje siguiendo la ruta recomendada es de: 0 days 00:08:53.666666666
 y la ruta recomendada es la siguiente:
```

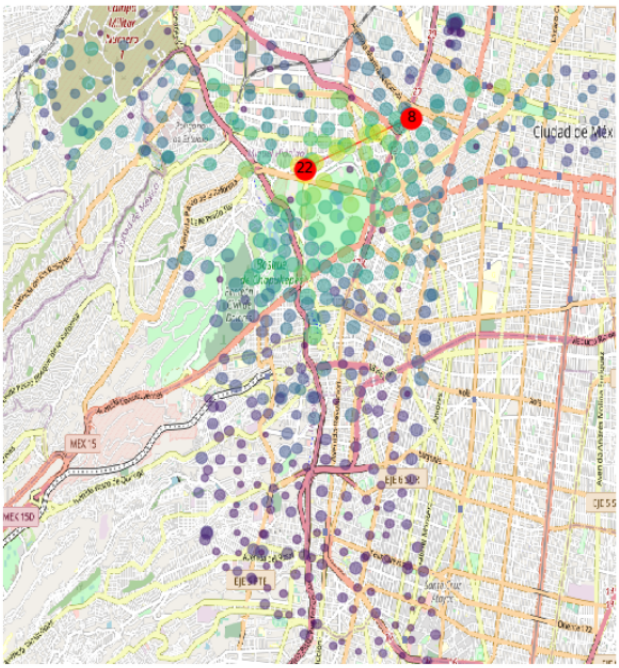

Figura 6: Tiempo estimado mas la ruta óptima de la estación 8 a la 22 a las 9 de la mañana

El tiempo estimado de tu viaje siguiendo la ruta recomendada es de: 0 days 00:09:47
y la ruta recomendada es la siguiente:

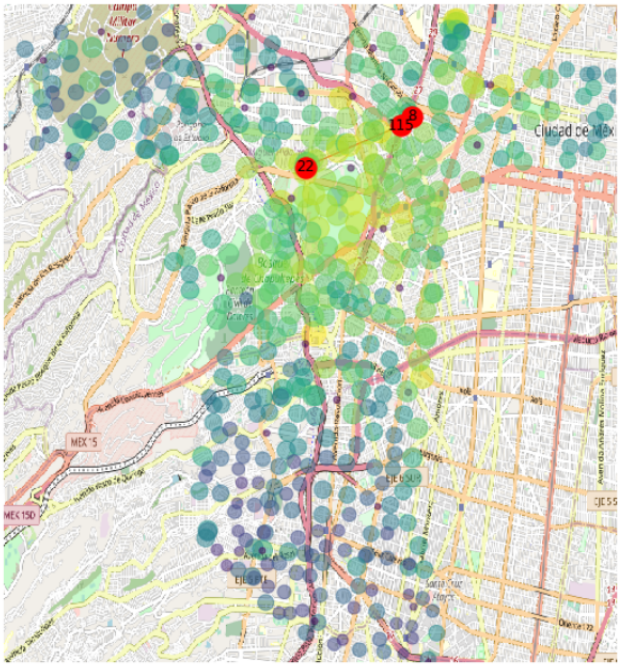

Figura 7: Tiempo estimado mas la ruta óptima de la estación 8 a la 22

De esto podemos concluir que es importante tomar en cuenta la manera en la que evoluciona el tráfico en la Ciudad de México a lo largo del día. Ya que no es lo mismo un trayecto en la mañana o en la tarde.

Conclusiones

Podemos ver bastantes cosas a partir de este análisis.
Notamos que hay una gran conexión entre las zonas que tienen este servicio, ya que todas las colonias se conectan con todas las demás, lo que nos hace ver que este es un sistema de transporte bastante bueno para llegar a diferentes colonias que utilicen ecobicis.
A contrario de lo que sospechábamos es que las estaciones que se encuentran en el centro geográfico son las que mayor centralidad tienen, en comparación con las periferias, sobre todo en el sur no hay tanto uso de eco-bicis como en las otras zonas.Esto se debe probablemente a que en el centro se encuentran zonas bastante concurridas en la Ciudad como es Reforma,Polanco y el Bosque de Chapultepec, además de que de aquí se puede partir o llegar de una forma más rápida a las otras regiones.
Por ultimo notamos que el trayecto en bici en estas zonas es bastante rápido, de hecho en los dos ejemplos que vimos te ahorras por lo menos la mitad del trayecto.Por lo que en efecto es un gran medio de transporte ya que no solo dejas de producir gases de tipo invernadero, sino haces ejercicio y además ahorras bastante tiempo.
Por ultimo creemos que es una buena opción aumentar las estaciones de ecobici que hay en la ciudad, claro haría falta otros estudios de relieve y de mercado para saber si de verdad es factible para diferentes áreas.Pero creemos que mientras mayores sean las estaciones de ecobici y a más lados puedas llegar con este servicio la demanda aumentará.
Finalmente cabe recalcar que no se tienen los datos suficientes para poder implementar el algoritmo

a distintas horas del día ya que al nuestro dataset ser muy pequeño hay ocasiones en los que no se tienen datos de trayectos entre dos estaciones a una hora específica. Del mismo modo sabemos que mientras más datos tengamos mas precisas pueden llegar a ser nuestras predicciones.

Documentación técnica del proyecto

Planteamiento del problema

Queremos saber el tiempo aproximado y la ruta óptima de cualquier trayecto del sistema de transporte ecobici,esto se hará con ayuda del algoritmo de Dijkarta.
Así mismo hacer un análisis rápido de el uso de este servicio por zonas para ver si es requerido aumentar las estaciones en las periferias.

Modelado matemático

Para tratar de resolver este problema utilizaremos el concepto de gráficas dirigidas y con pesos, las cuales constan de nodos los cuales serán las estaciones de ecobici y los enlaces dirigidos y ponderados, los cuales se generan cuando hubo un trayecto de una estación a otra y el peso de estos es el tiempo que duro el trayecto.
Para saber la concurrencia de cada estación vamos a utilizar la centralidad de grado y la de eigenvector.
La centralidad de eigenvector es una medida de la afluencia de un nodo de un gráfica, en donde toma en cuenta la conexión que tiene con otros nodos, sobre todo a los nodos que tienen una gran cantidad de vecinos que pocos.Matemáticamente viene dado como $Ax = \lambda x$ donde A es la matriz de adyacencia, con eigenvalor λ para el nodo i.[3]
Ahora la centralidad de grado esta normalizada ya que dividimos el total de vértices que tiene un nodo por la cantidad de vértices totales. Para saber el camino más rápido utilizaremos el algoritmo de Dijkstra, el cual explicare a más profundidad en el apartado siguiente.

Enunciado problemas algorítmicos

Ahora vamos a explicar que hace el algoritmo utilizado el cual es Dijkstra.
Lo que se encarga de hacer es encontrar el camino posible entre dos nodos el cual se minimice el peso que se va a usar.
Lo primero que tenemos es una gráfica dirigida con N nodos donde tomamos el nodo inicial i ,creamos un arreglo con distancias infinitas de i a cualquier otro nodo.
Ahora tomamos a $a = i$ en donde se recorren todos los nodos adyacentes de a excepto los nodos previamente marcado.A los nodos no marcados se les llamara v_i
Ahora para el nodo actual se calcula una distancia tentativa de este a sus vecinos, utilizando la siguiente formula $dt(v_i) = D_a + d(a, v_i)$ ya que tenemos la distancia del nodo a al inicial y ahora vemos la distancia de a a v_i, una vez hecho para todos los vecinos se marca a a como completo.
Ahora se toma como nodo actual al que tenga menor distancia en el arreglo de distancias y volvemos a iterar los pasos pasados, esto se hace hasta que no existan nodos marcados.

Para este algoritmo nos basaremos en la implementación para python que vimos en clase.[2]

```
#Implementamos Dijkstra
def dijkstra(G,u,v):
    masinf=float('inf')
    vertices=list(G.nodes)
    distancias={w:masinf for w in vertices}
```

```
    fijos={w:False for w in vertices}
    padres={w:None for w in vertices}
    distancias[u]=0
    fijos[u]=True
    nuevo_fijo=u

    while not(all(fijos.values())):
        # Acualizar distancias.
        for w in G.neighbors(nuevo_fijo):
            if fijos[w]==False:
                nueva_dist=distancias[nuevo_fijo]+G[nuevo_fijo][w]['weight']
                if distancias[w]>nueva_dist:
                    distancias[w]=nueva_dist
                    padres[w]=nuevo_fijo

        # Encontrar el nuevo a fijar.
        mas_chica=masinf
        for w in vertices:
            if fijos[w]==False and distancias[w]<mas_chica:
                optimo=w
                mas_chica=distancias[w]
        nuevo_fijo=optimo
        fijos[nuevo_fijo]=True

        # Cuando fije el v rtice final v, dar el camino.
        if nuevo_fijo==v:
            camino=[v]
            while camino[0]!=u:
                camino=[padres[camino[0]]]+camino
            return distancias[v], camino
```

Análisis de correctitud y análisis asintótico de tiempo y espacio

Aquí haremos un análisis de el algoritmo de Dijktra, vemos cual es su análisis asintótico de tiempo y de espacio, y la correctitud de esta implementación.
Primero nos vamos a centrar en el de tiempo, para eso hace falta notar que podemos expresar a la gráfica como una matriz de adyacencia .Ahora notamos que agregar a todos los vértices en la lista de fijos va a tomar $O(v)$ donde v es la cantidad de vértices, esto se debe a que se deben de hacer esta cantidad de comparaciones.
En otro paso vemos que encontrar los vértices con misma distancia también tarda $O(v)$ y el tiempo para recalcular la nueva distancia es $O(1)$,pero vemos que para hacer estas comparaciones, van dentro de un ciclo que también es de $O(v)$ que mientras se van fijando vértices .
Por lo que la complejidad de tiempo es de la forma

$$O(v) + O(v) * (O(v) + O(1)) = O(v^2)$$

Ahora nos fijamos en la de espacio.
Primero creamos una lista de todos los nodos de la grafica, también un arreglo de distancias con v entradas, uno de padres y otro de fijos también con v entradas.Por lo que la complejidad de espacio queda como.
Ahora notamos que la máxima cantidad de vértices que puede tener una gráfica dirigida es de $n(n-1)$ donde n es la cantidad de nodos.Por lo que la complejidad de espacio para Dijkstra tiene la siguiente

forma:

$$O(n) + 3O(n(n-1)) = O(n^2)$$

Por ultimo veremos la correctitud de este algoritmo.Para esto tendremos que ver que no se olvide ningún nodo o vértice y que en efecto te regresa el camino con menos peso posible.
Para eso notamos que le damos una gráfica y el nodo inicial al algoritmo.Nos fijamos en las distancias de todos sus vecinos y las almacenamos en una lista.Luego vemos la distancia de los vecinos de los vecinos que también guardamos en una lista, haciendo que esta distancia sea la distancia del padre más el peso de el vértice.Esto se hace hasta que se hayan revisado todos los nodos de la componente conexa, y con esto obtenemos la distancia mínima de el nodo inicial a cualquiera de la gráfica, hace falta notar que si un vértice o esta en la componente conexa del nodo inicial su valor que regresa es infinito ya que nunca va a poder llegar a el.
Por lo que con esto podemos decir que su implementación es correcta.

Aplicación a los datos concretos

Lo primero es importar los datos de las estaciones de ecobici, eliminamos las columnas que no nos interesan y perseguimos a graficar cada estación según sus coordenadas geográficas, viendo las 50 distintas colonias que tiene este servicio.
Ahora cargamos los datos de viajes que hubo de Enero del 2020 a Abril 2020, donde nos dice la edad y el género de usuario,la estación de retiro y de arribo junto con la fecha y hora de la cual se retiro y arribo.
Aquí notamos que utilizan algunas estación que no están registradas en ningún lado, suponemos que se hace referencia a que se las llevaron a hacer reparaciones o cosas por el estilo, así que las eliminamos.
Proseguimos a restar la hora de arribo menos la de retiro, para ver el tiempo total de viaje, este se convierte en valores numéricos para así poder ponderar los aristas .
Ahora vamos a hacer tres gráficas distintas, una en donde solo tenga datos de Enero para no tomar en cuenta el efecto Covid, la segunda en donde va a tener alrededor del 1 % de los datos totales esto,esto para poder dibujar una gráfica, de manera rápida y ver más o menos como es el comportamiento de esta, por ultimo creamos una gráfica con todos los datos del dataset, esto para tener la mejor ruta posible.
Hace falta notar que hay rutas que se repiten, por lo que vamos a sacar el tiempo promedio de estas rutas para sacar el peso de un solo enlace.
Ahora implementamos Dijkstra como se vio en (Enunciado problemas algorítmicos).
Creamos nuestra función en la cual pedimos la gráfica que vamos a usar, la estación de donde se quiere partir y a la cual se quiere llegar,proseguimos a llamar Dijkstra y el primer valor que nos da lo convertimos a timedelta, ya que estaba como numeric y este va a ser el tiempo que en promedio se tardara en llegar de tal estación a la otra.Creamos dos diccionarios uno en donde guardamos la centralidad de grado y otro la centralidad de eigenvector.Graficamos nuestra red en donde los nodos se encuentran en su posición geográfica y el tamaño viene dado por la centralidad de grado y el color por la de eigenvector.
Creamos otra red la cual tomara los nodos del camino más rápido encontrado por Dijkstra y sus vértices.
Por ultimo regresamos las dos gráficas donde podemos ver la ruta recomendada junto con el tiempo aproximado.
A continuación viene el código hecho en python.

```
#Creamos una funci n en donde le das la gr fica , tu nodo inicial y el final
#Utilizamos Dijkstra
#Vemos el peso total entre esos nodos y lo convertimos a formato de tiempo
#Mostramos la gr fica con la ruta  ptima  a seguir
#El  tama o de los nodos dice la centralidad de grado
#El color de los nodos indica la centralidad de eigenvector
def ruta(G,ini,fin):
  D=dijkstra(G,ini,fin)
  tiempo=pd.to_timedelta(D[0])
  masinf=float('inf')
  if D[0] !=masinf:
    print('El tiempo estimado de tu viaje siguiendo la ruta recomendada es de:',
   tiempo,'\n y la ruta recomendada es la siguiente:')

  plt.figure(figsize = (10,10))
  diccionario1 = nx.degree_centrality(G)
  sizes = np.array([diccionario1[i] for i in G])
  diccionario2 = nx.eigenvector_centrality(G)
  colors = np.array([diccionario2[i] for i in G])
  nx.draw_networkx_nodes(G,
                         node_size = 250*sizes,
                         #node_size=grados,
                         cmap = 'viridis',
                         node_color = colors,
                         pos=pos,
                         alpha=0.5)

  Gp = nx.Graph()
  nx.add_path(Gp, D[1])
  nx.draw(Gp,pos=pos,edge_color='r',node_color='r',with_labels=True)
  img = plt.imread('map.png')
  coor=((D2.longitude.min(),D2.longitude.max(),D2.latitude.min(),D2.latitude.max()
   ))
  plt.imshow(img, extent=coor, interpolation='nearest')
  plt.show()
```

Ahora veremos algunos ejemplos de la implementación de este algoritmo:

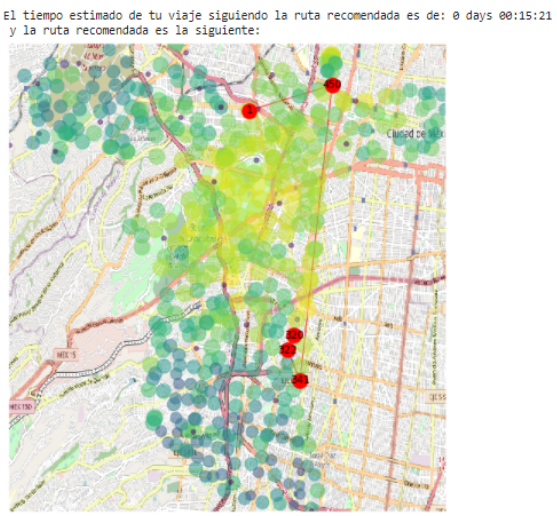

Figura 8: Tiempo estimado más la ruta recomendada de la estación 1 RIO SENA-RIO BALSAS a la 320 ENRIQUE RÉBSAMEN-LUZ SAVIÑON

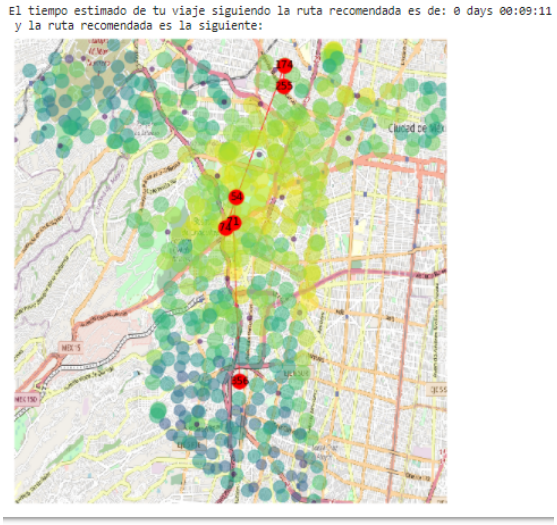

Figura 9: Tiempo estimado más la ruta recomendada de la estación 255 JOSE ROSAS MORENO-GUILLERMO PRIETO a la 356 MATIAS ROMERO -ADOLFO PRIETO

Al fijarnos en estas rutas de ejemplo parece ser que hace algunas vueltas son innecesarias, ya que por ejemplo en la ruta que va de la estación 1 a la 320 la cual queda al sur de esta, pero al principio se va para el norte.Con la información que tenemos no sabemos si de verdad esta es la ruta más rápida posible, pero también se puede deber a que no tenemos los datos suficientes para ver si existe una ruta alternativa y mas eficiente,una forma de arreglar esto sería ver mas datos históricos y con eso ya tendríamos mayor confianza de que nuestra ruta es la óptima.
Por otro lado vamos a comparar los tiempos que nos dice la ruta usando ecobici contra los tiempos que nos marca google maps para coche de la misma estación de origen y destino.
Vemos que usando ecobici para ir de Rió Sena a Enrique Rebsamen nos tardaríamos un aproximado de 15 minutos con 21 segundos, en comparación con lo checado en Google maps alrededor de las 6 de la tarde del 3 de Diciembre del 2021 que marca 31 minutos.Y viendo la ruta que va de José Rosas Moreno-Guillermo a Matias Romero, en ecobici nos marca que tardamos 9 minutos con 11 segundos, en cambio con el mismo método de la ruta anterior Google maps nos dice que en coche es alrededor de 34 minutos.
Hace falta sacar un promedio de cada estación i a cada estación j y comparar los tiempos que nos dice nuestro algoritmo en comparación con lo que dice google maps para autos.Pero parece ser que es mas eficiente y rápido moverse por estas zonas en bicicleta que en coche.

Ahora bien para implementar la predicción que tome en cuenta la hora del día, haremos algo similar a lo anterior, de hecho utilizaremos de nueva cuenta el algoritmo de Djikstra.
Lo primero que haremos será generar 24 gráficas dirigidas con pesos, estamos generando una por hora del día.
Lo una vez preprocesadas estas gráficas simplemente tenemos que utilizar nuestro algoritmo construido anteriormente pero especificaremos cuál de las 24 gráficas usar ya que cada una corresponde a una hora del día y por lo tanto usando el mismo algoritmo podemos mejorar o al menos obtener una perspectiva distinta a nuestro problema.

Apéndice

Algo que se intentó durante es proyecto es el siguiente enfoque para obtener la ruta más óptima considerando distintas horas del día, esto para darle diversidad a las técnicas utilizadas para atacar un mismo problema y aplicar los conocimientos adquiridos en el curso, la idea era calcular todas las rutas posibles entre dos vértices, esto por obvias razones es costoso en tiempo computacional sin embargo una vez obtenidos estos datos, en teoría sería fácil dados dos vértices obtener la ruta más corta a partir de todos los caminos haríamos un sort y podríamos obtener más fácil el de menor tiempo promedio.
La ventaja que tiene este método respecto al primer algoritmo sería que una vez teniendo la base de datos o el dataframe con todos los tiempos de todas las rutas posibles, el algoritmo para obtener la ruta más óptima dependería de la complejidad de nuestro ordenamiento.
Más aún es fácil ver que la base de datos no necesitaría ser actualizada de manera recurrente ya que debido a la ley de los grandes números eventualmente el tiempo promedio será el tiempo esperado, lo único realmente tardado de este método sería encontrar todos los caminos posibles entre todas las combinaciones de vértices posibles.
Primero veamos el algoritmo propuesto para esta tarea:

```
pat=[]
paths=[]
def FindAllPaths(inicio, final):
  if(len(pat)==0):
    pat.append(inicio)
  vecinos=[i for i in nx.all_neighbors(Gr, inicio)]
  for nextnode in vecinos:
    if(nextnode == final):
      temporary=[]
      for node1 in pat:
        temporary.append(node1)
      temporary.append(final)
      paths.append(temporary)
    elif(nextnode not in pat):
      pat.append(nextnode)
      FindAllPaths(nextnode, final)
      pat.pop()
```

Al ser este un intento fallido, no se profundizará en la correctitud del algoritmo.
Sin embargo podemos notar varias cosas, el algoritmo está basado en DFS.
Por otro lado nuestro algoritmo ignora ciclos.
Ahora bien, otra cosa que hay que destacar es que si nuestra gráfica tiene m aristas, entonces sucede que nuestro algoritmo tendría que calcular todos los caminos posibles entre dos vértices, y esto lo efectuará m veces con nodos distintos.
Por otro lado, en el peor de los casos nuestra gráfica será completa, entonces esto quiere decir que se tendrían que $n!$ llamadas recursivas a nuestra función.
Es decir por cada uno de las m aristas, se tendrían que hacer $n!$ llamadas a nuestra función.
Siguiendo con el peor de los casos al ser completa, se tendría que $m = \frac{n(n-1)}{2}$ entonces en esencia la complejidad computacional de nuestro algoritmo sería $O(\frac{n(n-1)}{2}n!) = O(n!)$
Considerando que tenemos alrededor de 480 estaciones de ecobicis, necesitaríamos que nuestro programa corriera $480!$ operaciones, y considerando que $19!$ es la edad aproximada del universo en segundos Fuente Por lo tanto nuestro algoritmo es una terrible idea, y ni hablar acerca del espacio

que se necesitaría para guardar dichos.
Finalmente podemos ver que efectivamente a medida que aumentamos la cantidad de nodos en nuestra gráfica el tiempo comienza a ser asintótico:

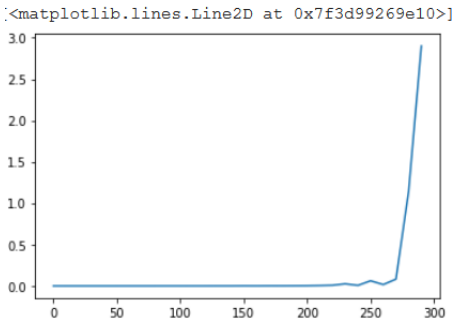

Figura 10: Tiempo de ejecución de nuestro algoritmo al tomar una muestra de tamaño $n \in (0, 320)$ (eje x), al principio no se tarda demasiado, ya que no hay muchas conexiones y la mayoría de los nodos sino es que todos tiene un único vecino, pero mientras vamos tomando muestras más grandes la cantidad de vecinos promedio aumenta y por lo tanto el tiempo se vuelve asintótico

Conclusiones

Como se puede ver al probar muchas estaciones de ecobici distantes entre si, el tiempo de trayecto es bastante corto, a pesar de que nuestro programa puede que no tenga la ruta más rápida (ya que hace vueltas innecesarias en el mundo físico),pero si podemos estar seguros que dada cierta red este encuentra el mejor camino siguiendo las aristas.Por lo que todavía hay mucho trabajo que hacer para calcular mejores tiempos y rutas(Posibles trabajos a futuro).
Por otro lado viendo los nodos más centralizados notamos que estos se encuentran en lo que viene siendo el centro geográfico de nuestra gráfica, por las zonas de Chapultepec, Reforma y Polanco.Lo cual tiene sentido ya que no solo son áreas con mucha afluencia de personas, sino que de estas se puede llegar fácilmente a las periferias de la red.

Posibles trabajos a futuro

Al usar los datos nos dimos cuenta de hay muchas cosas que se pueden mejorar o plantear nuevos proyectos con estos mismos.
Consideramos que la mejor forma de mejorar el algoritmo para calcular la mejor ruta optimizando tiempo de traslado es justo con más datos, ya que así la gráfica se conecta más por lo que los trayectos con sus tiempos serán más directos en vez de pasar por nodos que nos desvían del camino.Así mismo con más datos podemos disminuir el efecto que ocasionan datos atípicos como puede ser un ciclista que vaya muy rápido o al contrario alguno que vaya bastante despacio, encontrando la media de estos.
Algo que nos hubiera gustando mucho hacer es tratar a la gráfica como una red con flujos, en donde cada nodo tiene un limite de bicicletas que puede guardar, y ver como es que se va distribuyendo el uso de estas a lo largo del tiempo y del espacio , el problema es que las bases de datos que utilizamos no decía cuantas bicicletas había en cada estación dado un tiempo, por lo que no tenemos las condiciones iniciales para empezar con este proyecto, por lo que se podría buscar alguna base de datos que tenga esta información y a partir de esto implementarlo.

Referencias

1. Redacción (Septiembre 10, 2019), "¿Cuántas horas pierdes en el tráfico al año? Descúbrelo en este ranking por ciudad".El Financiero.https://www.elfinanciero.com.mx/nacional/sabes-cuantas-horas-pierdes-en-el-trafico-al-ano-mira-este-ranking-por-ciudad/

2. Leonardo Ignacio Martínez Sandoval.Consultado el 3 de Diciembre del 2021 http://madi.nekomath.com/P5/CaminosCortos.html

3. https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrali