Documentación técnica de Proyecto Final de Matemáticas Discretas

por Santiago Licea Becerril

Diciembre del 2022

Planteamiento del problema

Queremos obtener el camino cerrado en el que una persona, que llamaremos Juanito, pueda comprar sus productos del súper en menos de una hora (desde que sale de su casa hasta que llega) gastando la menor cantidad de dinero posible. Contamos con la ubicación de Juanito en coordenadas de latitud y longitud, así como la ubicación de cada una de las tiendas junto con los precios de todos los productos que se ofrecen. Buscamos que, dada una lista de productos y un tiempo máximo para realizar el recorrido de tiendas, podamos encontrar el mejor recorrido de tiendas para que se obtengan los productos al menor precio posible y terminemos antes del tiempo máximo.

Modelo matemático y suposiciones

En términos matemáticos, vemos que el problema se puede reducir a un problema algoritmico en gráficas. La forma más útil de visualizar a las tiendas y a Juanito es mediante un conjunto de vértices $V = \{V_1, V_2, ..., V_n\}$ en un plano. El problema también nos da un conjunto total de productos que denotaremos como $P = \{p_1, p_2, ..., p_k\}$. Asumiremos que para cada vértice, a excepción del cual se parte para hacer el recorrido (o sea Juanito), se tiene una lista de costos $C_i = \{c_1, c_2, ..., c_k\}$ donde el elemento $c_k \in C_i$ nos dice el costo del producto k en el vértice i.

Obtendremos un conjunto de productos $L \subset P$. Nosotros buscamos partir de un vértice V_i y encontrar un camino cerrado en el que se minimize una suma de costos de productos $\in L$ visitando vértices $\in V \setminus V_i$.

Buscaremos que este camino cerrado sea recorrido en menos de un tiempo $tmax$. Para este trabajo se hizo la suposición de que el tiempo que nos toma en ir de un vértice V_i a uno V_j es el mismo que de V_j a V_i. Dado que queremos conocer el tiempo que se toma en ir de cualquier vértice V_i a cualquier otro $V_j \in V \setminus V_i$, vemos que el problema se puede visualizar mediante una gráfica completa no

dirigida K_n con $n = |V|$.

Sabemos que una gráfica completa tiene $m = \frac{n(n-1)}{2}$ aristas, por lo que podemos definir a nuestro conjunto de aristas no dirigidas como:
$E = \{(V_1, V_2), (V_1, V_3), ..., (V_1, V_n), (V_2, V_3), (V_2, V_4), ..., (V_2, V_n), ..., (V_{n-1}, V_n)\}$
con pesos equivalentes a los tiempo de viaje $T = \{t_1, t_2, ..., t_m\}$ respectivamente. Así es como podemos ver a nuestro problema en términos abstractos.

Problema algorítmico a resolver

Tenemos una gráfica completa no dirigida K_n con pesos para cada arista. A excepción de un vértice V_i, cada vértice $V_j \in V \setminus V_i$ cuenta con costos diferentes para cada uno de los productos $P = \{p_1, p_2, ..., p_k\}$.

Dada una lista de productos $L \subset P$ y el vértice V_i, encontrar un camino cerrado que empiece y termine en V_i tal que se tomen los productos de L durante el recorrido, se minimice la suma de costos de cada producto, y la suma de los pesos de las aristas no exceda un límite $tmax$.

Propuesta de solución

Veamos que si $|L| = 1$, podemos recorrer un camino cerrado en el que solo visitemos un vértice aparte del V_i inicial y final para obtener el producto deseado. Si visitamos más vértices, por principio de las casillas, habrá vértices en los que no tomemos ningún producto. En general, si $|L| = l$, buscaremos recorrer l o menos vértices. Veamos que si $|L| = 5$, podemos visitar 5 vértices en donde "compremos" un producto en cada uno, pero también podemos solo visitar 2 vértices y obtener 3 productos en uno y 2 en otro.

Notemos que, dado que nosotros buscamos caminos cerrados en donde no se repitan vértices a excepción del vértice inicial, nos estamos enfocando en los ciclos de V_i a V_i en nuestra gráfica K_n. Cabe aclarar que estamos denotando al vértice $V_i \in V$ como el vértice del que se parte para hacer el ciclo. Este vértice no cuenta con una lista de costos de productos.

En el ejemplo anterior asumimos que hay más vértices que número de productos en L. No obstante, si buscamos 5 productos pero solo hay 2 vértices aparte del V_i, entonces solo podemos visitar 2 o menos vértices.

Por simplicidad, denotemos a los ciclos como "viajes", a $|V|$ como $numVertices$, y a $|L|$ como $numProductos$. Veamos que el número de viajes posibles para la obtención de los productos es una función de $numVertices$ y de $numProductos$. Si consideramos $numVertices = 5$ y $numProductos = 3$, veamos que podemos decidir, partiendo desde V_i, solo visitar uno de los vértices restantes y obtener ahí todos los productos. En tal caso solo existen $numVertices - 1 = 4$ viajes posibles desde V_i. Si decidimos visitar 2 vértices, veamos que existen $\binom{4}{2}$ viajes

posibles. Aquí asumimos que el viaje V_i, V_j, V_k, V_i es igual a V_i, V_k, V_j, V_i. Por último, si decidimos visitar 3 vértices, veamos que existen $\binom{4}{3}$ viajes posibles.

De aquí se deduce la fórmula

$$numViajes = \sum_{i=1}^{numProductos} \binom{numVertices}{i}$$

si $numProductos \leq numVertices$

o bien

$$numViajes = \sum_{i=1}^{numVertices} \binom{numVertices}{i}$$

si $numProductos > numVertices$

Podemos considerar a la cardinalidad del espacio de estados donde está la solución como el numero de viajes que se obtiene con esta fórmula. Encontrar el viaje con peso total menor a $tmax$ y en el que se gaste lo menos posible por los productos calculando todos los viajes posibles y al final elegir el que cumpla con estas caraterísticas suena poco óptimo. Suena incluso menos óptimo si consideramos que para cada viaje habría que calcular todas las formas en que se pueden tomar los productos.

En vez de hacer una búsqueda exhaustiva partiendo desde cero, decidimos usar la información que ya tenemos. Si solo nos enfocamos en un producto $p_k \in L$, sabemos que el costo de este en un vértice V_j es el elemento $c_k \in C_j$. Si recabamos todos los elementos de la forma c_k de los vértices $V_j \in V \setminus V_i$, obtenemos una lista de los diferentes costos para el producto p_k que denotaremos como $X_k = (x_1, x_2, ..., x_n)$ donde el elemento $x_n \in X_k$ es el costo del producto k en el vértice V_n. En total tendremos $numProductos$ de estas listas, una para cada producto.

La solución propuesta radica en ordenar a las listas X_k que correspondan a los productos en L de menor a mayor. Por ejemplo, si $L = \{p_1, p_3, p_{10}\}$, entonces debemos ordenar las listas X_1, X_3, y X_{10}. Si obtenemos que la lista X_3 ordenada es $X_3 = (x_8, x_2, x_5, ...)$, esto nos dice que en el vértice V_8 se encuentra el precio más bajo para el producto p_3.

Nuestro algoritmo empezará tomando el primer elemento de cada lista X_k y fijandose en qué vértices se encuentran dichos costos. Calcularemos el peso del ciclo que empieze en V_1, pase por esos vértices, regrese a V_1, y veremos si es menor o igual a $tmax$. Si esto último se cumple, entonces esta será la solución a nuestro problema. Habremos encontrado el camino que nos cueste menos de $tmax$ en el que obtengamos el costo más barato por nuestros productos.

No obstante, debemos fijarnos en qué pasa si la suma del peso de las aristas de este ciclo es mayor a $tmax$. En ese caso, queremos que nuestro algoritmo tome el segundo mejor costo de alguna de las listas X_k (el cual corresponde a otro vértice) y vuelva a evaluar el peso del camino. Volviendo al ejemplo anterior, supongamos que las listas obtenidas son $X_1 = (x_7, x_6, x_9, ...)$, $X_3 = (x_8, x_2, x_5, ...)$, y $X_{10} = (x_3, x_{12}, x_1, ...)$. Esto quiere decir que para el producto p_1 los mejores vértices por visitar son $(7, 6, 9, ...)$, para el producto p_3: $(8, 2, 5, ...)$, y para el producto p_{10}: $(3, 12, 1, ...)$. Si nuestro primer recorrido de vértices $(V_1, V_7, V_8, V_3, V_1)$ tiene un peso mayor que $tmax$, entonces nuestra segunda opción es el ciclo $(V_1, V_7, V_8, V_{12}, V_1)$.

Cuando se acaben los vértices del producto p_{10} (es decir, cuando recorramos $numVertices$ de posibilidades), cambiaremos ahora al segundo mejor vértice del producto p_3 y volveremos a iterar sobre la lista de los vértices del producto p_{10}.

Ahora bien, se hizo un último ajuste a este algortimo. Dado que estamos siguiendo un orden en los bucles iterativos, debemos evaluar cuál es el mejor orden posible de los productos para minimizar el precio. Asumimos que, si tenemos que para los productos p_1, p_3, y p_{10} tenemos los precios mínimos de cada uno dados por $55, 20, 200$ respectivamente (y sabemos en qué vértices están), entonces nos conviene ordenar a los productos de tal forma que en el nivel más alto de los bucles anidados esté el producto con el precio mínimo más alto y en el nivel más bajo esté el producto con el precio mínimo más bajo. Asumimos que, para el producto p_{10}, la diferencia de su costo mínimo de 200 menos el segundo mejor costo para ese producto es mayor a la diferencia de costo del producto p_3 con precio mínimo de 20 menos el segundo mejor costo de este producto.

Por ello, si nuestro primer recorrido pesa más de $tmax$, nuestra segunda opción será elegir la segunda mejor opción de costo del producto cuyo costo mínimo es el más pequeño de los costos mínimos de los productos que estemos buscando.

Análisis asintótico y de correctitud

La solución propuesta está muy ligada al contexto del problema. En el ajuste recién explicado en donde definimos un orden para los bucles, asumimos una mayor variabilidad del costo para productos con mayor precio. Es decir, si el precio más barato de un producto es de 200 y el precio más barato de otro es de 50, asumimos que 200 menos el segundo precio mínimo para ese producto es mayor a 50 menos el segundo precio mínimo para ese producto. Esto suele darse en productos reales, teniendo como ejemplo el caso extremo en que un producto cueste 1000 y otro 4.5.

Incluso en el mundo real puede que esto no siempre se cumpla. Matemáticamente, no se dieron especificaciones de las distribuciones de los costos de cada producto, por lo que ordenar los bucles no nos aseguraría encontrar la suma de costos mínima posible. Se ordenaron los bucles para no tener que calcular todas las

permutaciones del orden de estos y ver cuál genera la solución con la suma de costos mínima. Esto es una de las limitaciones de la solución. No obstante, para el contexto en el que surgió este problema abstracto, creemos que es una solución pertinente dada las distribuciones de los costos de productos alimenticios.

Se hizo una segunda suposición ligada al contexto real del problema. Asumimos que en la mayoría de los casos no tendremos que recorrer todos los bucles al hacer la búsqueda exhaustiva. A diferencia de una búsqueda exhaustiva, nosotros tenemos la ventaja de que en cualquier momento de la búsqueda podemos encontrar nuestra solución ya que estamos recorriendo el espacio de estados en un orden que nos garantiza que al encontrar un ciclo con peso menor a $tmax$, entonces habremos encontrado la solución.

Esto lo podemos interpretar como la parte voraz de nuestro algoritmo. De todo el espacio de ciclos posibles directamente tomamos aquel que optimice el costo de cada uno de los productos que buscamos y calculamos el peso de este ciclo. No tuvimos que calcular todos los ciclos posibles, almacenar los pesos y costos para cada uno y al final ver cuál es el que toma menos de $tmax$ y el que tenga el costo más bajo por los productos.

No obstante, puede darse el caso de que los vértices con los menores costos estén muy alejados del vértice V_1 y solo unos cuantos vértices cuyo costo por los productos es alto estén cerca y estos sean los que correspondan a la única solución. En este caso, tendremos que recorrer todos los bucles.

Dado que debemos considerar el peor caso para el análisis de complejidad, veamos que si tomamos a $numVertices$ como n y a $numProductos$ como k, tendremos una compledidad de tiempo de $O(n^k)$ ya que para cada producto tendremos un bucle anidado en el que tendremos que recorrer $numVertices$ elementos. Para la complejidad de espacio, tendremos que almacenar los vértices con la lista de costos para cada uno y las aristas con sus respectivos pesos. Esto da una complejidad en espacio de $O(kn + \frac{n(n-1)}{2}(2)) = O(kn + n^2)$.

Algoritmos y estructuras de datos empleadas

Se empleó un data frame para almacenar el nombre de cada producto, los respectivos costos, y el vértice al que corresponden. Después se particionó este conjunto en data frames para cada producto y se llevó a cabo el ordenamiento de menos a más respecto a los costos de cada uno. Para almacenar las aristas y los pesos, se empleó un diccionario G en donde la llave corresponde al nombre de la arista en formato $(V1, V2)$ y el valor corresponde al peso.

Cabe destacar que gracias al diccionario G, se recabaron los pesos de las aristas en tiempo constante. Esto fue muy importante dado que solo almacenamos $\frac{n(n-1)}{2}$ aristas, pero si llegásemos a recorrer todos los ciclos posibles, en el peor de los casos tendriamos que recorrer un número de aristas:

$$numAristasMax = \sum_{i=1}^{numProductos} (i+1)\binom{numVertices}{i}$$

si $numProductos \leq numVertices$

o

$$numAristasMax = \sum_{i=1}^{numVertices} (i+1)\binom{numVertices}{i}$$

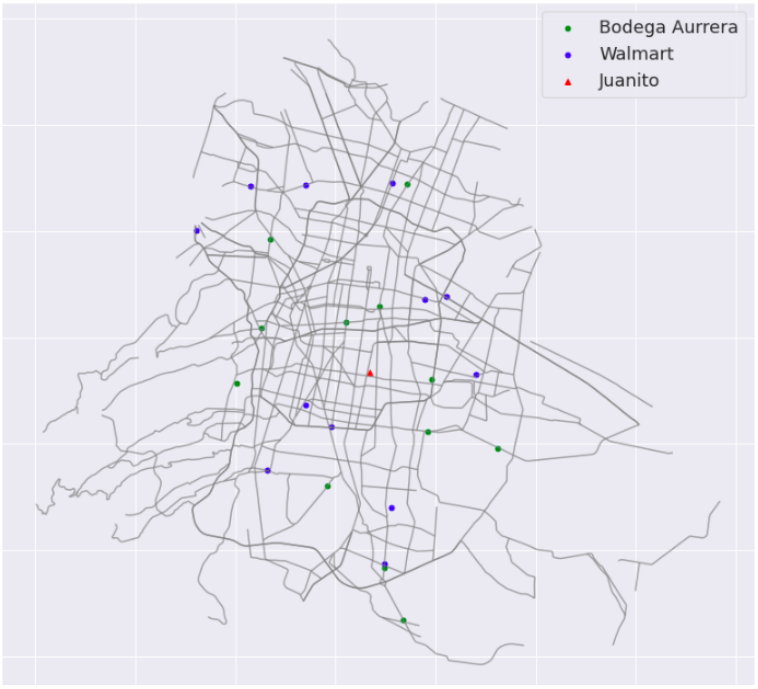
si $numProductos > numVertices$

ya que muchos ciclos se intersectan dentro de la gráfica completa.

Aplicación a los datos concretos

Para el conjunto de datos Quien es Quien en los Precios se filtraron los datos para acotar el problema a la Ciudad de México y por motivos de demanda computacional solo consideramos las tiendas Walmart y Bodega Aurrera. El conjunto de datos solo incluye una muestra de cada una de estas cadenas comerciales, por lo que en total solo nos quedamos con 24 tiendas. Decidimos buscar 3 productos y dar un $tmax = 60$. Todas estas acotaciones se pueden modificar en el programa de Python ya que se manejaron como variables que al modificarse en su declaración inicial se puede hacer que el resultado obtenido se reajuste.

La limpieza y transformación de los datos se detalla en los comentarios del notebook. A continuación enseñamos una visualización del problema que hicimos en un mapa de la CDMX con ayuda de geopandas usando los datos que filtramos y dando una ubicación para el vértice V_i = Juanito:



Los vértices los nombramos como coordenadas (latitud, longitud). Por lo que las llaves del diccionario (que corresponden a las aristas no dirigidas) quedaron de la forma (latitudX,longitudX,latitudY,longitudY). Los pesos de las aristas (o sea los tiempos de viaje) los obtuvimos con ayuda de una API llamada OpenStreetMap. Veamos que si buscamos el tiempo entre las aristas (latitudX,longitudX) y (latitudY,longitudY), podemos buscar en nuestro diccionario la llave (latitudX,longitudX,latitudY,longitudY) y si no está entonces buscamos (latitudY,longitudY,latitudX,longitudX).

Como al iterar sobre los bucles se puede dar el caso de que tengamos que calcular el tiempo de una ruta en la que se repita un vértice, al buscar (latitudX,longitudX,latitudX,longitudX) en el diccionario no encontraremos nada por lo que no añadiremos ningún tiempo de viaje.

Veamos un ejemplo de la solución que dio nuestro algoritmo a una entrada en particular. Para la ubicación de Juanito que se mostró en el mapa, decidimos buscar los productos:

```
[ ] prods = ['1 KG. GRANEL. CARNE PARA ASAR', 'BOTE 900 GR. FRESA', 'PAQUETE C/12. BLANCO']
```

que corresponden a presentaciones de carne, yoghurt, y huevo. Haciendo la llamada a la función:

```
[ ] tiempo, precio, prod1Coord, prod2Coord, prod3Coord = voraz(prod1, prod2, prod3, coordsJuanito, G, 60)
```

donde prod1, prod2, prod3 corresponden a los data frames ordenados por costo de cada producto mencionados en la sección de "Algoritmos y estructuras de datos empleadas", coordsJuanito son las coordenadas que definimos para Juanito, G el diccionario de aristas y pesos, y 60 es el número máximo de minutos que deseamos tardarnos en nuestro reccorido de tiendas. Obtuvimos:

```
[ ]  tiempo, precio, prod1Coord, prod2Coord, prod3Coord

     (59.5475,
      227.0,
      '19.407087,-99.144805',
      '19.357925,-99.152347',
      '19.368269,-99.164848')
```

Obtuvimos que nos tardaremos 59.54 minutos (menos de 60 minutos), pagaremos $227 por nuestros productos (el precio más barato para el límite de tiempo que establecimos), y tenemos que visitar las tiendas cuyas coordenadas de latitud y longitud son las que se muestran en la imagen.

Algo que se puede mejorar de este despliegue de información es que no nos muestra algo fundamental que es qué productos comprar en qué tienda. Esto se puede solucionar fácilmente, pero por lo mientras, la aplicación nos muestra el orden en que debemos recorrer las tiendas donde la primera ubicación que muestra corresponde a la tienda en la que deberemos obtener el producto por el que pagaremos la mayor cantidad de dinero y la última tienda corresponde a la tienda en la que obtendremos el producto por el cual pagaremos la menor cantidad de dinero (por el orden de los bucles que explicamos en la sección de "Propuesta de solución").

Conclusiones

En conclusión, llegamos a una solución para nuestro problema algorítmico que, aunque no es del todo óptima si se considera dentro de un contexto puramente computacional, sí que resuelve el problema si consideramos los datos con los que trabajamos por motivos mencionados en el análisis de correctitud. Aparte de esto, este trabajo tiene muchas otras mejoras y propuestas de trabajos futuros. Podríamos mejorar la forma en la que se despliega la respuesta para darnos más información como la visualización de la ruta u otras posibles opciones que también sean óptimas.

Con nuestro algoritmo podríamos hacer un estudio de en qué zonas de la ciudad le cuesta más a la gente comprar una lista del super dentro de un mismo tiempo límite. Podríamos no solo acotar nuestros datos a productos alimenticios y a cadenas comerciales de alimentos, para poder admitir una lista de productos más variada.

Finalmente, sería un proyecto muy interesante poder llevar esto a una aplicación real para los mexicanos en donde se considere un conjunto de datos más grande (ya que con el que trabajamos solo incluye una muestra de las sucursales de algunas cadenas comerciales) y que se actualice diariamente para todas las tiendas de todos los estados del país. Existe una aplicación y una página web en donde se hizo uso de este conjunto de datos para ayudar a los mexicanos a tomar decisiones. Desafortunadamente, a día de hoy ambas implementaciones no funcionan correctamente.

Bibliografía

- Profeco. (2022). Quién es Quién en los precios. Datos Abiertos de México. Recuperado de: https://datos.gob.mx/busca/dataset/quien-es-quien-en-los-precios
- (n,a). (2022). Vialidades de la Ciudad de México. Datos Abiertos de México. Recuperado de: https://datos.cdmx.gob.mx/dataset/vialidades-de-la-ciudad-de-mexico/resource/f246ca7d-ff19-472f-9cf3-556e55c4e34a
- Dekanovsky, V. (2020). Driving distance between two or more places in python. Towards Data Science. Recuperado de: https://towardsdatascience.com/driving-distance-between-two-or-more-places-in-python-89779d69-1def
- Stewart, R. (2018). GeoPandas 101: Plot any data with a latitude and longitude on a map. Towards Data Science. Recuperado de: https://towardsdatascience.com/geopandas-101-plot-any-data-with-a-latitude-and-longitude-on-a-map-98e01944b972