

## *Dijkstra: Optimización de rutas aéreas y emisiones de Co2.*

*Grupo 7 Algoritmos*

*Santiago Cassiano Rozo, Juliana Catalina De Castro Moreno, Luis Fernando Mendez Marques, Ivan Alexander Morales Munoz, Juan Esteban Oviedo García, Nicolás Mauricio Rincón Vija.*

Departamento de ingeniería de sistemas y computación  
Universidad Nacional de Colombia  
Bogotá, Colombia

[scassiano@unal.edu.co](mailto:scassiano@unal.edu.co)  
[jdec@unal.edu.co](mailto:jdec@unal.edu.co)  
[lumendezm@unal.edu.co](mailto:lumendezm@unal.edu.co)  
[imorales@unal.edu.co](mailto:imorales@unal.edu.co)  
[joviedog@unal.edu.co](mailto:joviedog@unal.edu.co)  
[nrinconv@unal.edu.co](mailto:nrinconv@unal.edu.co)

**Abstract**—In this project we created a Python program that identifies and graphs the routes between two cities in which the distance, CO2 emissions or Emissions/Distance ratio of flights between those cities will be minimized. To do this, we represent the cities as nodes and the flights as edges in a graph and apply Dijkstra's algorithm to find the path in which the above parameters are minimized. Using the program we find that depending on the parameter to be minimized, the optimal path between the two cities may be different.

**Keywords**—Algorithm; Dijkstra; Flights; Optimization; CO2; Distance.

### I. INTRODUCCIÓN

La optimización de trayectorias, conlleva al constante desarrollo de múltiples herramientas y estudios computacionales, evaluando series de casos cuya complejidad crece constantemente gracias a la necesidad natural del ser humano para conectarse con las personas. Estos estudios parten de modelos sencillos para definir el problema que tienen como resolución, modelos más complejos cuyas estadísticas son imprescindibles a lo largo del tiempo. Estos modelos permiten obtener resultados que como primera aproximación pueden ser de mucha utilidad, especialmente a nivel cualitativo. Los dos factores entre varios posibles que se tendrán en cuenta en este proyecto a la hora de estudiar las rutas óptimas son las ciudades, en concreto las distancias que yacen entre ellas, y la huella de carbono subyacente de estos viajes. Principalmente, nuestro primer enfoque causa que la ruta óptima siga caminos diferentes y nuestro segundo enfoque es un impacto global

del cual no podemos hacer caso omiso. Así pues, la trayectoria óptima desde una mirada heurística resulta en un mayor grado de realismo.

Como mencionamos anteriormente, la creciente preocupación internacional por las consecuencias adversas del cambio climático ha impulsado a las organizaciones e instituciones a profundizar su conocimiento respecto de los gases de efecto invernadero y su comportamiento. En este contexto, la huella de carbono se transforma en un indicador reconocido internacionalmente para comprender dicha dinámica [1], lo que implica ir más allá de la moción industrial, no solo conocerla en todas su dimensión, sino que medirla y divulgarla como un elemento más en los procesos de toma de decisiones individuales, de las empresas, regiones o países. En este caso nos enfocaremos en la huella de carbono que tienen los aviones por medio de diferentes distancias dadas en vuelos internacionales.

### II. PROBLEMA

Hoy en día, la globalización se ha convertido en algo cada vez más común. La necesidad que se tiene de transportarse no solo al interior de un territorio, sino también alrededor del planeta es bastante significativa. Con base en lo anterior, debido a la demanda, los precios de este tipo de viajes han aumentado considerablemente en los últimos años, además, no existen rutas que aseguren la conexión entre todos los destinos posibles, y tampoco una conexión directa.

A modo de ejemplo, es imposible trasladarse entre Bogotá y Melbourne sin realizar múltiples escalas. Así, surge la necesidad de evaluar opciones que existen para realizar este tipo de trayectos de larga distancia, en principio, haciendo una optimización de la distancia, así mismo, con las emisiones de CO2 dado un determinado vuelo.

Por otro lado, en razón a la contaminación global, es fundamental reducir el impacto medioambiental de cada vuelo, por este motivo también es necesario buscar una optimización de las emisiones de CO2 en relación a la distancia (CO2/km) .

### III. OBJETIVOS

- Emplear el algoritmo de Dijkstra para realizar los ajustes necesarios a un grafo cuyos nodos actúan como puntos de ciudades en el mapa y cuyas

aristas representan las distancias y emisiones de CO<sub>2</sub> entre estas ciudades (Para aquellas que sí se encuentran conectadas) con el propósito de buscar siempre la mejor ruta de un destino a otro.

- Construir, a partir de los datos de distancias y emisiones, visualizaciones de los grafos generados a partir de los datos recolectados previamente.
- Permitir al usuario final evidenciar los trayectos propuestos por el algoritmo para su desplazamiento entre las dos ciudades propuestas, incluyendo no solo la lista de ciudades a recorrer, sino también sus ubicaciones en el mapa.

#### IV. MARCO TEÓRICO

Para el desarrollo de este proyecto haremos énfasis en el algoritmo de Dijkstra con sus respectivas modificaciones, donde tomaremos el problema como un grafo en el que todos los aeropuertos son nodos y los vuelos son las aristas con el peso del intervalo de distancia, es decir, la distancia del aeropuerto de destino y la distancia del aeropuerto de origen.

##### A. Dijkstra

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Se trata de una especialización de la búsqueda de costo uniforme, por lo cual no funciona en grafos con aristas de coste negativo.

Pseudo-pasos:

1. Marca el nodo inicial que elegiste con una distancia actual de 0 y el resto con infinito.
2. Establece el nodo no visitado con la menor distancia actual como el nodo actual A.
3. Para cada vecino V de tu nodo actual A: suma la distancia actual de A con el peso de la arista que conecta a A con V. Si el resultado es menor que la distancia actual de V, establecese como la nueva distancia actual de V.
4. Marca el nodo actual A como visitado.
5. Si hay nodos no visitados, ve al paso 2.

Ejemplo:

Calcularemos la distancia más corta entre el nodo C y los demás nodos del grafo [2] :

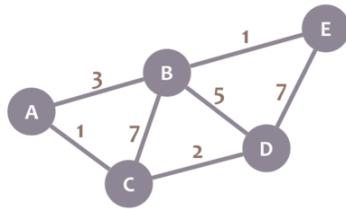


Figura 1.

Durante la ejecución del algoritmo, iremos marcando cada nodo con su *distancia mínima* al nodo C.

Para el nodo C, esta distancia es 0. Para el resto de nodos, como todavía no conocemos esa distancia mínima, empieza siendo infinita ( $\infty$ ). En la imagen, marcaremos el nodo actual con un punto rojo:

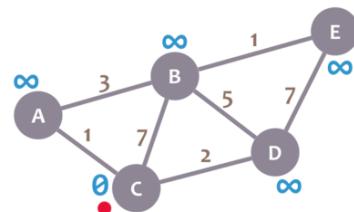


Figura 2.

Ahora, revisaremos los vecinos de nuestro nodo actual (A, B y D) en cualquier orden. Empecemos con B. Sumamos la mínima distancia del nodo actual (en este caso, 0) con el peso de la arista que conecta al nodo actual con B (en este caso, 7), y obtenemos  $0 + 7 = 7$ . Comparamos ese valor con la mínima distancia de B (infinito); el valor más pequeño es el que queda como la distancia mínima de B (en este caso, 7 es menor que infinito):

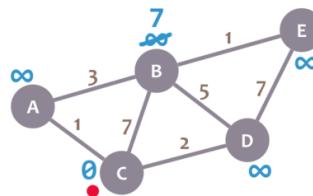


Figura 3.

Ahora revisaremos al vecino A. Sumamos 0 (la distancia mínima de C, nuestro nodo actual) con 1 (el peso de la arista que conecta nuestro nodo actual con A) para obtener 1. Comparamos ese 1 con la mínima distancia de A (infinito) y dejamos el menor valor:

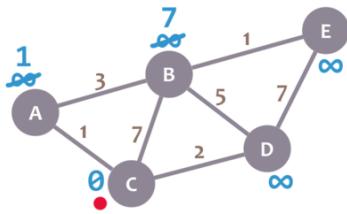


Figura 4.

Repetimos el procedimiento para D:

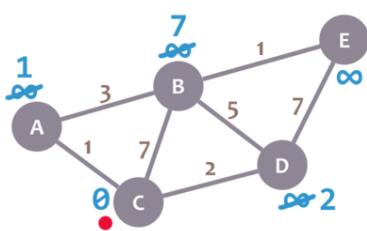


Figura 5.

Marcamos como C como *visitado*. Representemos a los nodos visitados con una marca de verificación verde:

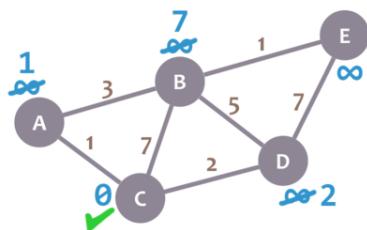


Figura 6.

Ahora debemos seleccionar un nuevo *nodo actual*. Ese nodo debe ser el nodo no visitado con la menor distancia mínima, es decir, el nodo con el menor número y sin marca de verificación verde. En este caso, ese nodo es A. Vamos a marcarlo con el punto rojo:

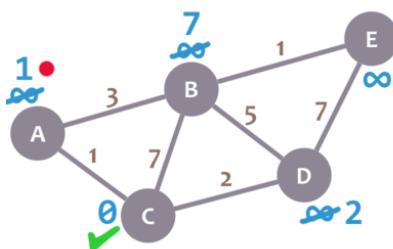


Figura 7.

Ahora, repetimos el algoritmo. Revisamos los vecinos de nuestro nodo actual, ignorando los visitados. Esto significa que solo revisaremos B.

Para B, sumamos 1 (la distancia mínima de A, nuestro nodo actual) con 3 (el peso de la arista conectando a A con B) para obtener 4. Comparamos ese 4 con la distancia mínima de B (7) y dejamos el menor valor: 4.

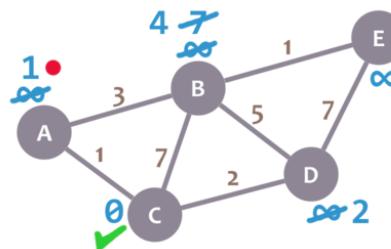


Figura 8.

Después, marcamos A como visitado y elegimos un nuevo nodo: D, que es el nodo no visitado con la menor distancia mínima.

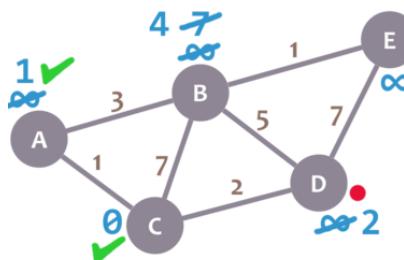


Figura 9.

Repetimos el algoritmo de nuevo. Esta vez, revisamos B y E.

Para B, obtenemos  $2 + 5 = 7$ . Comparamos ese valor con la distancia mínima de B (4) y dejamos el menor valor (4). Para E, obtenemos  $2 + 7 = 9$ , lo comparamos con la distancia mínima de E (infinito) y dejamos el valor menor (9).

Marcamos D como visitado y establecemos nuestro nodo actual en B.

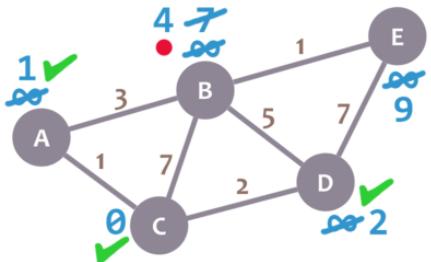


Figura 10.

Debemos verificar  $E. 4 + 1 = 5$ , que es menos que la distancia mínima de E (9), así que dejamos el 5. Marcamos B como visitado y establecemos E como el nodo actual.

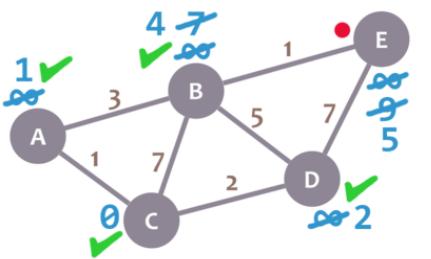


Figura 11.

E no tiene vecinos no visitados, así que no verificamos nada. Lo marcamos como visitado.

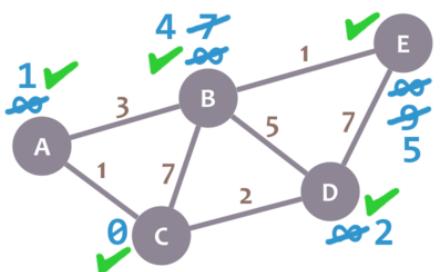


Figura 12.

Como no hay nodos no visitados, terminamos. La distancia mínima de cada nodo ahora representa la mínima distancia entre ese nodo y el nodo C, es decir, el nodo que elegimos como nodo inicial.

## B. Cola de prioridades del algoritmo de Dijkstra

En el algoritmo de Dijkstra, partimos de un nodo fuente e inicializamos su distancia en cero. A continuación, empujamos el nodo origen a una cola de prioridad con un coste igual a cero. A continuación, realizamos varios pasos.

En cada paso, extraemos el nodo con el menor coste, actualizamos las distancias de sus vecinos y los empujamos a la cola de prioridad si es necesario.

Cada uno de los nodos vecinos se inserta con su respectivo nuevo coste, que es igual al coste del nodo extraído más la arista que acabamos de atravesar. Seguimos visitando todos los nodos hasta que no haya más nodos que extraer de la cola de prioridad.

## C. Librerías utilizadas e instalación de paquetes del proyecto:

- geopy.
- pandas.
- numpy.
- NetworkX.
- cartopy.

**geopy:** Esta librería de Python, facilita la localización de las coordenadas de direcciones, ciudades, países y puntos de referencia en todo el mundo utilizando geocodificadores de terceros y otras fuentes de datos. [3]

**pandas:** Es una biblioteca de software escrita como extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. [4]

**numpy:** Es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. [5]

**NetworkX:** Es una biblioteca de Python para el estudio de grafos y análisis de redes. [6]

**cartopy:** Cartopy es un paquete de Python diseñado para el procesamiento de datos geoespaciales con el fin de producir mapas y otros análisis de datos geoespaciales, especialmente resulta útil para los datos de gran superficie y pequeña escala, en los que tradicionalmente se rompen los supuestos cartesianos de los datos esféricos. [7]

## D. Emisiones de Co2.

En base a los datos de [emisiones promedio de un vuelo](#) se efectúa el cálculo para los orígenes y destinos mencionados previamente. Para esto tendremos en cuenta algunos factores que explicaremos a continuación.

Sin embargo, debemos considerar un factor imprescindible, y es que el presupuesto total de carbono restante a nivel mundial es de 400.000 millones de toneladas de CO2 a partir de enero de 2020, según los acuerdos de París y otros acuerdos. [8]

Por lo tanto, esto se traduce en solamente 50 toneladas de CO<sub>2</sub> por persona como límite de por vida, lo cual es bastante alarmante.

En los países con altas emisiones, este presupuesto personal de carbono se agotará en pocos años, por ejemplo, en 3 años a finales de 2024 para el Reino Unido. [9]

Así que nos podemos preguntar, ¿La aviación es sostenible? Porque si bien, los vuelos de ocio no son asequibles dentro de los presupuestos de carbono. La mayoría de los gobiernos, empresas y ciudadanos niegan o engañan deliberadamente sobre la urgencia de la emergencia climática y la necesidad de cambios radicales en los estilos de vida.

A partir de nuestro contexto anterior podemos mencionar las bases para los cálculos que necesitamos, Ergo:

1. Base 1 para el cálculo: a partir del consumo de combustible por vuelo

Una forma de calcular las emisiones de CO<sub>2</sub> es a partir del consumo de combustible por vuelo.

Entonces tenemos que un avión Boeing 737 - 400 suele utilizarse para vuelos internacionales cortos. Entonces, para una distancia de 926 km, se calcula que la cantidad de combustible utilizada es de 3.61 toneladas , incluyendo el rodaje, el despegue, el crucero y el aterrizaje. Esto se realiza por medio del siguiente análisis, si se utiliza una capacidad de 164 asientos y una ocupación media de los asientos o "factor de carga" del 65% , se obtiene un consumo de combustible de 36.6 g por pasajero-kilómetro. Así mismo, las emisiones de CO<sub>2</sub> del combustible de aviación son de 3,15 gramos por gramo de combustible, por lo cual obtenemos emisiones de CO<sub>2</sub> para un Boeing 737 - 400 de 115 g por pasajero km. También cabe resaltar que a una velocidad de crucero de 780 km por hora, esto equivale a 90 kg de CO<sub>2</sub> por hora.

En cifras correspondientes para un Boeing 747 - 400 utilizado para vuelos internacionales de larga distancia obtenemos:

- Distancia: 5556 km
- Combustible utilizado: 59.6 toneladas
- Asientos: 416
- Ocupación de los asientos: 80 %.
- Número medio de pasajeros: 333
- Consumo de combustible por pasajero km: 59,6 toneladas / (5556 km x 333) = 32,2 g por pasajero km
- Emisiones de CO<sub>2</sub>: 101 g por pasajero km (multiplicando por 3,15 g de CO<sub>2</sub> por g de combustible)
- Velocidad de crucero: 910 km por hora
- Emisiones de CO<sub>2</sub> 92 kg de CO<sub>2</sub> por hora

Así pues, para ambos aviones, las emisiones son de unos 90 kg de CO<sub>2</sub> por hora.

Este CO<sub>2</sub> se emite generalmente en la alta atmósfera, y se cree que tiene un mayor efecto invernadero que el CO<sub>2</sub> liberado a nivel del mar. Por lo tanto, las emisiones se ajustan multiplicándose por un factor de 2,00 (véase "Forzamiento radiativo" más adelante) para obtener 180 kg de CO<sub>2</sub> equivalente por hora.

También hay que tener en cuenta la energía de los combustibles fósiles utilizada en :

- La extracción y el transporte de petróleo crudo
- La ineficacia de las refinerías (alrededor del 7%)
- La fabricación y el mantenimiento de las aeronaves y la formación del personal
- La construcción, el mantenimiento, la calefacción, el alumbrado de los aeropuertos, etc.

Por lo tanto, las emisiones de CO<sub>2</sub> se redondean y la calculadora de Carbono Independent toma un valor de 250 kg, es decir, 1/4 de tonelada de CO<sub>2</sub> equivalente por hora de vuelo.

## V. METODOLOGIA

### 1. Lectura del dataset “geolocation.csv”

Este dataset contiene los datos de la latitud y longitud de las 31 ciudades elegidas en el proyecto para poder graficarlas.

Además ajustamos los nombres de las columnas en los datos recibidos de “geolocation.csv”. Por lo tanto tendremos de manera intuitiva: CountryName, CapitalName, CapitalLatitude, CapitalLongitude, CountryCode, ContinentName. Y por último mostramos los datos obtenidos de las ciudades.

A continuación el código:

```
from geopy import distance
import pandas as pd
import numpy as np
import math

data = pd.read_csv("geolocation.csv")

data.rename(columns={"CountryName": "Country", "CapitalName": "capital", "CapitalLatitude": "lat", "CapitalLongitude": "lon", "CountryCode": "code", "ContinentName": "continent"}, inplace=True)

data
return go(f, seed, [])
```

Figura 13.

### 2. Lectura del dataset “KM.csv”

El dataset “KM.csv” contiene la información de la distancia de los vuelos directos entre las ciudades elegidas para el proyectos las cuales son:

Bogota, Madrid, Munich, Londres, Amsterdam, Istanbul, Nueva York, Tókio, São Paulo, Fortaleza, Paris, Roma, Rio

de Janeiro, Helsinki, Doha, Ciudad del cabo, Montreal, Toronto, Miami, Hangzhou, Beijing, Lima, Cancún, Buenos Aires, Hong Kong, Sidney, Manama, New Delhi, Zúrich, Ciudad de México, Estocolmo.

Posteriormente mostramos los datos leídos en el dataset “KM.csv”. En nuestro código, aparecerán filas y columnas las cuales respectivamente representarán, la ciudad de la que parte el vuelo y la ciudad a la que llega.

Para resaltar una distancia de 0, indica que no hay vuelos directos.

A continuación el código:



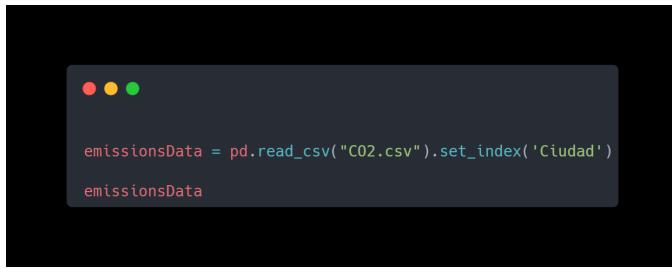
```
● ● ●  
distanceData = pd.read_csv("KM.csv").set_index('Ciudad')  
distanceData
```

Figura 14.

### 3. Lectura del dataset “Co2.csv”

Este dataset contiene la información de la emisión en Kg de Co2 de los vuelos directos entre las ciudades elegidas para el proyecto las cuales ya se mencionaron anteriormente. De manera paralela como en nuestro dataset anterior mostramos los archivos leídos, posicionamos la fila como indicativo de la ciudad de la que parte el vuelo, y la columna la ciudad a la que llega el vuelo.

A continuación el código:



```
● ● ●  
emissionsData = pd.read_csv("C02.csv").set_index('Ciudad')  
emissionsData
```

Figura 15.

### 4. Lectura del dataset “CP.csv”

Este dataset contiene la información del cálculo de la relación Emisiones/Distancia (CO2/KM) de los vuelos directos entre las ciudades elegidas para el proyecto.

Mostramos los datos leídos del archivo "CP.csv". La fila indica la ciudad de la que parte el vuelo, y la columna la ciudad a la que llega.

Dato para resaltar en este caso, una cantidad de 10000 indica que no existe el vuelo directo.

A continuación el código:



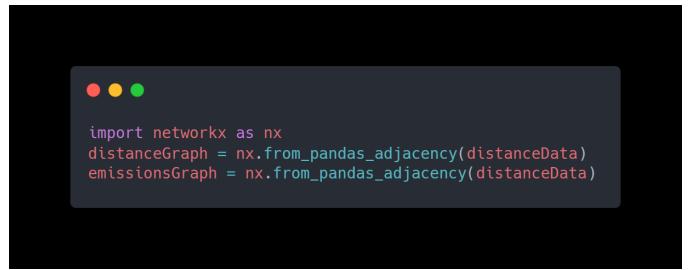
```
● ● ●  
globalData = pd.read_csv("CP.csv", decimal=",").set_index('Ciudad').astype(float)  
globalData
```

Figura 16.

### E. Visualización del grafo.

Para la visualización del grafo se emplea la librería NetworkX y graficamos con matplotlib.

A continuación el código:



```
● ● ●  
import networkx as nx  
distanceGraph = nx.from_pandas_adjacency(distanceData)  
emissionsGraph = nx.from_pandas_adjacency(emissionsData)
```

Figura 17.

Posteriormente vamos a dividir nuestro análisis en dos partes.

El primero será el gráfico de distancias, para esto importamos el graficador matplotlib y ajustamos los colores, etiquetas y el tamaño de la figura.



```
● ● ●  
import matplotlib.pyplot as plt  
nx.draw_circular(distanceGraph, node_color="#42f59b", with_labels=True, edge_color="#074710",  
label="Graph Representation for Cities Dataframe", style='dashed')  
plt.figure(figsize=(30,30))  
plt.show()  
print('\n\n\n')
```

Figura 18.

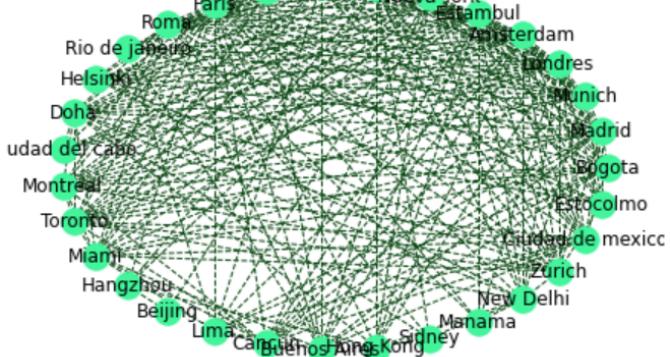


Figura 19.

Para el gráfico de emisiones realizamos de manera similar nuestro anterior paso.



Figura 20.

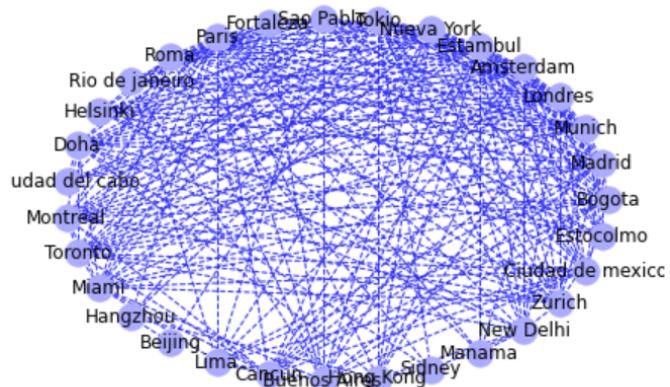


Figura 21.

Ahora bien, si analizamos la figura 19 y 21 encontramos que en los gráficos no existe una diferencia significativa porque ambos se centran tanto en la distancia como las emisiones que se obtienen de ciudad en ciudad por lo cual lo podemos definir en términos de una función sobreyectiva.

D. Creación del grafo utilizando una estructura propia.

#### 1. Definición de la estructura del grafo.

Crearemos un diccionario con todos los posibles nodos el cual se llamará “self.edges” y una tupla que tendrá todos los pesos entre cada nodo, la cual se llamará “self.weights”. Es

importante resaltar que los bordes serán bidireccionales. A continuación se muestra en código.

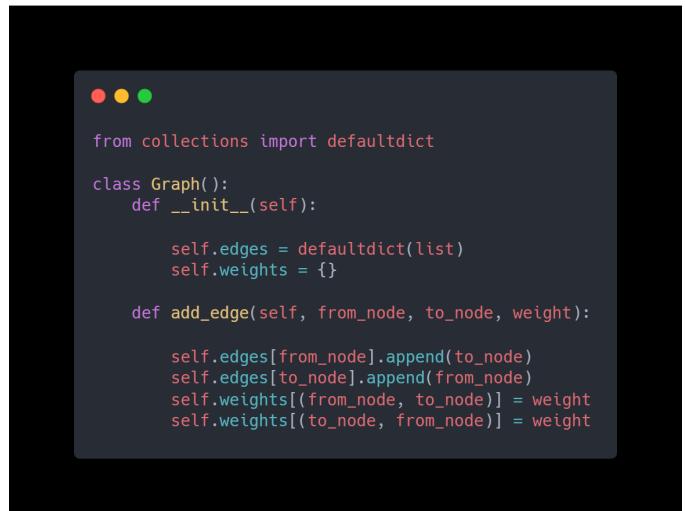


Figura 22.

Ahora colocamos las distancias en el diccionario, “distanceDict” e iteraremos sobre nuestro dataset “distanceData” en cada fila.



Figura 23.

De manera similar colocamos las emisiones en el diccionario “emissionsDict” e iteraremos sobre nuestro dataset “distanceData” en cada fila.



Figura 24.

Por último colocamos relación entre las emisiones y la distancia en Km en el diccionario “parameterDict” e iteramos sobre nuestro dataset “distanceData” en cada fila.

```

parameterDict = {}
for row in range(distanceData.shape[0]):
    org = cities[row]
    parameterDict[org] = {}
    for col in range(distanceData.shape[1]):
        dst = cities[col]
        if globalData.iloc[row][col] != 0 and globalData.iloc[row][col] != 10000:
            parameterDict[org][dst] = globalData.iloc[row][col]

```

Figura 25.

Finalmente al grafo agregaremos las ciudades teniendo en cuenta la distancia dada por “distanceGraph”, las emisiones dado por “emissionsGraph”, y la relación entre ambas dado por “parameterGraph”.

```

distanceGraph = Graph()
for origin in distanceDict.keys():
    for destination in distanceDict[origin]:
        if distanceDict[origin][destination] != None and distanceDict[origin][destination] != np.nan and distanceDict[origin][destination] != 0:
            distanceGraph.add_edge(origin, destination, distanceDict[origin][destination])

emissionsGraph = Graph()
for origin in emissionsDict.keys():
    for destination in emissionsDict[origin]:
        if emissionsDict[origin][destination] != None and emissionsDict[origin][destination] != np.nan and emissionsDict[origin][destination] != 0:
            emissionsGraph.add_edge(origin, destination, emissionsDict[origin][destination])

parameterGraph = Graph()
for origin in parameterDict.keys():
    for destination in parameterDict[origin]:
        if parameterDict[origin][destination] != None and parameterDict[origin][destination] != np.nan and parameterDict[origin][destination] != 10000:
            parameterGraph.add_edge(origin, destination, parameterDict[origin][destination])

```

Figura 26.

#### E. Dijkstra para encontrar el path más corto.

Previamente dimos los conceptos preliminares para acercarnos a un contexto, Ergo vamos a aplicar este algoritmo para encontrar la distancia más corta en vuelos teniendo en cuenta un punto de origen y un punto de destino. Dicho esto los parámetros de nuestra función principal serán, el grafo, un punto inicial y un punto final. Ahora bien el camino más corto será un diccionario de nodos cuyo valor es una tupla que comprende el nodo anterior y el peso. principalmente tendremos un ciclo while que evaluará si el nodo actual es diferente a nuestro punto final, si esto es así, cada nodo que visite ahora será nuestro nodo actual de tal manera que vaya recorriendo los caminos del grafo, sin embargo para ejecutar ese paso, tendrá en cuenta el peso entre los nodos y los irá comparando para así continuar cumpliendo con la premisa de obtener el camino más corto.

```

def dijkstra(graph, initial, end):
    shortest_paths = {initial: (None, 0)}
    current_node = initial
    visited = set()
    while current_node != end:
        visited.add(current_node)
        destinations = graph.edges[current_node]
        weight_to_current_node = shortest_paths[current_node][1]
        for next_node in destinations:
            weight = graph.weights[(current_node, next_node)] + weight_to_current_node
            if next_node not in shortest_paths:
                shortest_paths[next_node] = (current_node, weight)
            else:
                current_shortest_weight = shortest_paths[next_node][1]
                if current_shortest_weight > weight:
                    shortest_paths[next_node] = (current_node, weight)
        next_destinations = {node: shortest_paths[node] for node in shortest_paths if node not in visited}
        if not next_destinations:
            return "Route Not Possible"
        current_node = min(next_destinations, key=lambda k: next_destinations[k][1])
    path = []
    while current_node is not None:
        path.append(current_node)
        current_node = shortest_paths[current_node][0]
    current_node = end
    path = path[::-1]
    return path

```

Figura 27.

#### F. Ingreso de la ciudad de origen y destino del vuelo y ejecución de los algoritmos.

Este paso es uno de los puntos más importantes del algoritmo aquí ingresamos las ciudad de origen y destino. A partir de esto tendremos 3 funcionalidades, la primera será calcular la ruta para reducir la distancia de vuelo entre la ciudad de origen y de destino con “distanceResult”. La segunda será calcular la ruta para reducir las emisiones de vuelo entre la ciudad de origen y de destino con “emissionsResult” y la tercera será calcular la ruta para reducir la relación Emisiones/Distancia (CO2/KM) de los vuelos entre la ciudad de origen y de destino con “combinedResult”.

```

origen = 'Bogota'
destino = 'Sidney'

distanceResult = dijkstra(distanceGraph, origen, destino)
emissionsResult = dijkstra(emissionsGraph, origen, destino)
combinedResult = dijkstra(parameterGraph, origen, destino)

```

Figura 27.

#### G. Análisis de complejidad del algoritmo.

Dada la estructura de la implementación realizada por el equipo, el programa únicamente se encuentra parametrizado bajo un solo elemento que afecta la complejidad del algoritmo, el cual es la cantidad de ciudades incluidas en los datasets, es decir, la cantidad de nodos que tendrá cada uno de los grafos en los cuales buscaremos el camino más corto.

La complejidad del algoritmo utilizando la metodología Big O partiendo de la base de dicho parámetro anteriormente mencionado (cantidad de ciudades) se podría considerar como  $O(n)^2$ , donde el factor “n” es la cantidad de ciudades establecidas en base a la información condensada en los datasets. Esta complejidad está determinada por dos procesos que son los más complejos en nuestro programa, a continuación explicaremos cuales son:

- En primer lugar, construir cada uno de los grafos en los cuales cada nodo es una de las ciudades, y cada arista es un vuelo directo que hay entre dos ciudades. Para construir cada uno de dichos grafos iteramos sobre nuestros conjuntos de datos, los cuales tienen  $n^2$  datos, siendo “n” la cantidad de ciudades, por lo cual la complejidad de construir cada grafo es  $O(n)^2$
- En segundo lugar, el otro proceso lógico más complejo y determinante del programa es realizar el algoritmo de Dijkstra para cada uno de los grafos, de manera que se encuentre el camino de menor peso para ir de una ciudad a otra en los tres grafos que tenemos. La implementación que utilizamos del algoritmo de Dijkstra tiene una complejidad de  $O(n)^2$ , siendo n la cantidad de ciudades que tenemos en el grafo. Esta complejidad se obtiene debido a que se itera por una de las n ciudades en el grafo, y por cada ciudad, se revisa cada una de las aristas con las que se conecta con las otras ciudades en el grafo, lo cual en el peor de los casos podrían ser n-1 aristas.

Los dos procesos anteriores son los que tienen un valor de complejidad más alto en el programa y determinan la complejidad del mismo por medio de la expresión  $O(n)^2 + O(n)^2 = O(n)^2$ , siendo n la cantidad de ciudades que se utilizan como nodos en el programa.

#### H. Evaluación del algoritmo vs. librerías preestablecidas

Existe una librería llamada “Dijkstar” [10] para Python, la cual permite crear los grafos y aplicarles un algoritmo para encontrar el camino más corto entre dos nodos.

Es por ello que decidimos comparar el tiempo que tarda en ejecutarse nuestro programa para generar los grafos y ejecutar nuestra implementación del algoritmo de Dijkstra en cada uno versus el tiempo que tarda el mismo proceso empleando las herramientas que ofrece “Dijkstar”. A continuación mostramos la diferencia en segundos de la ejecución utilizando la librería Time de Python:

Número de Ejecución	Algoritmo Implementado (s)	Dijkstar (s)
1	0.02581	0.00477
2	0.01246	0.00469
3	0.01782	0.00628
4	0.01426	0.00986
5	0.0163	0.00511
6	0.01916	0.0045
7	0.01379	0.00508
8	0.01868	0.00468
9	0.01908	0.00468
10	0.02583	0.01278

Como se puede observar, la librería “Dijkstar” tarda mucho menos en ejecutar los procesos del algoritmo, seguramente esto se debe a que dicha librería utiliza estructuras de datos como heaps que permiten optimizar el tiempo de ejecución del algoritmo.

## VI. RESULTADOS

### 1. Mapa de las distancias:

El mapa de las distancias está dado con base al dataframe de las ciudades con sus latitudes y longitudes, entonces iteramos con un index sobre las ciudades con un contador para así obtener todas y agregarlas a los diccionarios “mapNames” y “mapIndexes”.



```

mapIndexes = {}
mapNames = {}
i = 0
for index, row in data.iterrows():
    if row["capital"] in distanceResult:
        mapIndexes[i] = [row["lat"], row["lon"]]
        mapNames[row["capital"]] = [row["lat"], row["lon"]]
    i += 1

mapNames = sortDictionary(mapNames, distanceResult)
mapIndexes = sortDictionaryIdx(mapIndexes, mapNames, distanceResult)

```

Figura 28.

Previamente escogimos como origen a Bogota y como destino a Sidney, por lo cual tendremos múltiples escalas que están representadas como ['Bogotá', 'Ciudad de mexico', 'Tokio', 'Sidney'] y gráficamente se ve así

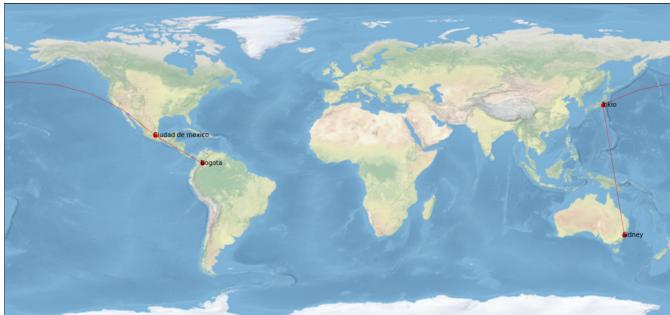


Figura 29.

## 2. Mapa de las emisiones.

El mapa de las emisiones está dado con base al dataframe de las ciudades con sus latitudes y longitudes, entonces iteramos con un index sobre las ciudades con un contador para así obtener todas y agregarlas a los diccionarios “mapNamesE” y “mapIndexesE”.



Figura 30.

Cuando graficamos en este caso el correspondiente análisis se da con que ahora obtenemos una nueva ruta la cual nos indica la ruta que nos ayuda a reducir la cantidad de emisiones, esta ruta está dada por ['Bogota', 'Paris', 'New Delhi', 'Sidney']



Figura 31.

## 3. Mapa de emisiones vs. Distancia.

El mapa de esta comparación está dado con base al dataframe de las ciudades con sus latitudes y longitudes, entonces iteramos con un index sobre las ciudades con un contador para así obtener todas y agregarlas a los diccionarios “mapNamesC” y “mapIndexesC”.



Figura 32.

Como mencionamos anteriormente este mapa es una comparación por lo cual si se quiere minimizar las emisiones de CO2 y la Distancia (CO2/KM) de los vuelos para ir de Bogotá a Sidney se debe tomar la ruta ['Bogotá', 'Londres', 'New Delhi', 'Sidney']

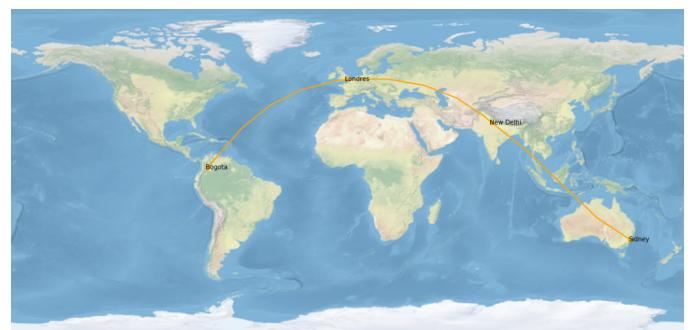


Figura 33.

## VII. CONCLUSIONES

- Al obtener los resultados se evidenció que los parámetros, en este caso las emisiones del avión en el trayecto, la distancia que recorre el avión y la relación entre las emisiones y la distancia, afectan de manera directa la decisión sobre qué ruta es la más óptima según el parámetro a optimizar que se elija.
- Viendo cómo se afecta una ruta dependiendo del parámetro elegido, a futuro se puede plantear

incrementar el número de parámetros, por ejemplo agregar precio, tiempo de vuelo, combustible, opciones de entretenimiento, entre otros. Con esto se pueden lograr hacer más relaciones entre los parámetros para ver cuál sería la ruta más óptima según los parámetros que se especifiquen.

- C. A partir de los parámetros se puede empezar a analizar cuáles serían las mejores opciones de vuelo y de esta manera eliminar aquellos que la gente no compra, así las aerolíneas pueden evitar hacer los vuelos que no se llenen y poder generar nuevas rutas más óptimas para los viajeros.

## VIII. REFERENCIAS

- [1] Samaniego, J. and Schneider, H., 2010. *La huella del carbono en la producción, distribución y consumo de bienes y servicios*. [online] Repositorio.cepal.org. Available at: <[https://repositorio.cepal.org/bitstream/handle/11362/3753/S2009834\\_es.pdf](https://repositorio.cepal.org/bitstream/handle/11362/3753/S2009834_es.pdf)> [Accessed 12 June 2022].
- [2] CodinGame. n.d. *Coding Games and Programming Challenges to Code Better*. [online] Available at: <<https://www.codingame.com/playgrounds/7656/los-caminos-mas-co>>
- [3] Geopy.readthedocs.io. 2022. *Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation*. [online] Available at: <<https://geopy.readthedocs.io/en/stable/>> [Accessed 12 June 2022].
- [4] Pandas.pydata.org. 2022. *pandas - Python Data Analysis Library*. [online] Available at: <<https://pandas.pydata.org/>> [Accessed 12 June 2022].
- [5] Numpy.org. 2022. *NumPy*. [online] Available at: <<https://numpy.org/>> [Accessed 12 June 2022].
- [6] Networkx.org. 2022. *NetworkX — NetworkX documentation*. [online] Available at: <<https://networkx.org/>> [Accessed 12 June 2022].
- [7] Scitools.org.uk. 2022. *Introduction — cartopy 0.20.0 documentation*. [online] Available at: <<https://scitools.org.uk/cartopy/docs/latest/>> [Accessed 12 June 2022].
- [8] Cusp.ac.uk. 2022. [online] Available at: <<https://cusp.ac.uk/wp-content/uploads/WP-29-Zero-Carbon-Sooner-update.pdf>> [Accessed 12 June 2022].
- [9] Carbonindependent.org. 2022. [online] Available at: <[https://www.carbonindependent.org/files/LCR\\_Tyndall\\_Centre\\_carbon\\_budget\\_report.pdf](https://www.carbonindependent.org/files/LCR_Tyndall_Centre_carbon_budget_report.pdf)> [Accessed 12 June 2022].
- [10] Dijkstar. 2022. Pypi. [online] Available at: <https://pypi.org/project/Dijkstar/>