**Dataset Correction**

**import** pandas **as** pd

**import** numpy **as** np

**import** math

**import** csv

columns **=** ['Flight No.', 'Timestamp', 'Altitude', 'Latitude', 'Longitude']

*# Step 1: Read and preprocess the dataset*

dataset **=** pd**.**read\_csv("NA\_11\_Jun\_29\_2018\_UTC11.CSV", sep **=** " ", names **=** columns)

dataset**.**to\_csv("NA\_11\_Jun\_29\_2018\_UTC11\_Output.CSV", index **=** **False**)

*# Extract the necessary columns: Flight No., Timestamp, Altitude, Latitude, and Longitude*

print(dataset**.**head(10))

**Creating dataset for transmission link**

data\_trans **=** {"Mode k":[1, 2, 3, 4, 5, 6, 7],

"Mode Color":["Red", "Orange", "Yellow", "Green", "Blue", "Pink", "Purple"],

"Switching Threshold(km)":[500, 400, 300, 190, 90, 35, 5.56],

"transmission Rate": [31.895, 43.505, 52.857, 63.970, 77.071, 93.854, 119.130]}

df\_trans **=** pd**.**DataFrame(data\_trans)

df\_trans**.**to\_csv("Transmission Link.csv", index **=** **False**)

print(df\_trans)

**Conversion to 3D Cartesian Coordinate and finding the Distance Between them**

**def** calculate\_distance(lat1, lon1, alt1, lat2, lon2, alt2):

earth\_radius **=** 6371 *# Radius of the Earth in kilometers*

lat1\_rad **=** math**.**radians(lat1)

lon1\_rad **=** math**.**radians(lon1)

lat2\_rad **=** math**.**radians(lat2)

lon2\_rad **=** math**.**radians(lon2)

equatorial\_radius **=** 6378.137 *# kilometers*

polar\_radius **=** 6356.752 *# kilometers*

eccentricity **=** math**.**sqrt(1 **-** (polar\_radius **\*\*** 2) **/** (equatorial\_radius **\*\*** 2))

x1 **=** (equatorial\_radius **+** alt1) **\*** math**.**cos(lat1\_rad) **\*** math**.**cos(lon1\_rad)

y1 **=** (equatorial\_radius **+** alt1) **\*** math**.**cos(lat1\_rad) **\*** math**.**sin(lon1\_rad)

z1 **=** ((equatorial\_radius **\*** (1 **-** eccentricity **\*\*** 2)) **+** alt1) **\*** math**.**sin(lat1\_rad)

x2 **=** (equatorial\_radius **+** alt2) **\*** math**.**cos(lat2\_rad) **\*** math**.**cos(lon2\_rad)

y2 **=** (equatorial\_radius **+** alt2) **\*** math**.**cos(lat2\_rad) **\*** math**.**sin(lon2\_rad)

z2 **=** ((equatorial\_radius **\*** (1 **-** eccentricity **\*\*** 2)) **+** alt2) **\*** math**.**sin(lat2\_rad)

distance **=** math**.**sqrt((x2 **-** x1) **\*\*** 2 **+** (y2 **-** y1) **\*\*** 2 **+** (z2 **-** z1) **\*\*** 2)

**return** distance

**Conversion to 3D Cartesian Coordinates for single value**

**def** Convert\_3D(latitude, longitude, altitude):

earth\_radius **=** 6371 *# Radius of the Earth in kilometers*

latitude\_rad **=** math**.**radians(latitude)

longitude\_rad **=** math**.**radians(longitude)

equatorial\_radius **=** 6378.137 *# kilometers*

polar\_radius **=** 6356.752 *# kilometers*

eccentricity **=** math**.**sqrt(1 **-** (polar\_radius **\*\*** 2) **/** (equatorial\_radius **\*\*** 2))

*# Heathrow Airport coordinates to x, y, z coordinates*

x **=** (equatorial\_radius **+** altitude) **\*** math**.**cos(latitude\_rad) **\*** math**.**cos(longitude\_rad)

y **=** (equatorial\_radius **+** altitude) **\*** math**.**cos(latitude\_rad) **\*** math**.**sin(longitude\_rad)

z **=** ((equatorial\_radius **\*** (1 **-** eccentricity **\*\*** 2)) **+** altitude) **\*** math**.**sin(latitude\_rad)

**return** x, y, z

**To find transmission rate**

**Function to calculate the data transmission rate based on the distance between airplanes**

**def** calculate\_transmission\_rate(distance):

**if** 500 **<=** distance:

**return** 31.895

**elif** 400 **<=** distance **<** 500:

**return** 43.505

**elif** 300 **<=** distance **<** 400:

**return** 52.857

**elif** 190 **<=** distance **<** 300:

**return** 63.970

**elif** 90 **<=** distance **<** 190:

**return** 77.071

**elif** 35 **<=** distance **<** 90:

**return** 93.854

**elif** 5.56 **<=** distance **<** 35:

**return** 119.130

**else**:

**return** 0

**Single Objective Optimization**

*# Load and preprocess the dataset*

airplanes **=** []

ground\_stations **=** []

flight\_name**=**input("Enter starting string code of flight: ")

**with** open('NA\_11\_Jun\_29\_2018\_UTC11\_Output.csv', 'r') **as** file:

reader **=** csv**.**reader(file)

next(reader) *# Skip the header row*

**for** row **in** reader:

flight\_no **=** row[0]

altitude **=** float(row[2])

latitude **=** float(row[3])

longitude **=** float(row[4])

*# Convert latitude, longitude, and altitude to 3D Cartesian coordinates*

coordinates **=** Convert\_3D(longitude, latitude, altitude)

x **=** coordinates[0]

y **=** coordinates[1]

z **=** coordinates[2]

**if** flight\_no**.**startswith(flight\_name):

airplanes**.**append((flight\_no, x, y, z))

**else**:

ground\_stations**.**append((flight\_no, x, y, z))

*# if flight\_no.startswith('AA'):*

*# airplanes.append((flight\_no, x, y, z))*

*# else:*

*# ground\_stations.append((flight\_no, x, y, z))*

*# if flight\_no.startswith('BA'):*

*# airplanes.append((flight\_no, x, y, z))*

*# else:*

*# ground\_stations.append((flight\_no, x, y, z))*

*# if flight\_no.startswith('DA'):*

*# airplanes.append((flight\_no, x, y, z))*

*# else:*

*# ground\_stations.append((flight\_no, x, y, z))*

*# if flight\_no.startswith('LH'):*

*# airplanes.append((flight\_no, x, y, z))*

*# else:*

*# ground\_stations.append((flight\_no, x, y, z))*

*# if flight\_no.startswith('UA'):*

*# airplanes.append((flight\_no, x, y, z))*

*# else:*

*# ground\_stations.append((flight\_no, x, y, z))*

**def** find\_max\_data\_rate\_routing\_paths(airplanes, ground\_stations):

routing\_paths **=** []

**for** airplane **in** airplanes:

airplane\_id, x\_airplane, y\_airplane, z\_airplane **=** airplane

max\_data\_rate **=** 0.0

max\_data\_rate\_path **=** []

**for** i **in** range(len(ground\_stations)):

**for** j **in** range(i **+** 1, len(ground\_stations)):

ground\_station\_id, x\_gs, y\_gs, z\_gs **=** ground\_stations[i]

next\_ground\_station\_id, next\_x\_gs, next\_y\_gs, next\_z\_gs **=** ground\_stations[j]

*# Calculate the distance between the current ground station and the next ground station*

distance **=** calculate\_distance(x\_gs, y\_gs, z\_gs, next\_x\_gs, next\_y\_gs, next\_z\_gs)

*# print(distance)*

*# Calculate the data transmission rate for the link*

transmission\_rate **=** calculate\_transmission\_rate(distance)

**if** transmission\_rate **>** max\_data\_rate:

max\_data\_rate **=** transmission\_rate

max\_data\_rate\_path **=** [(ground\_station\_id, transmission\_rate), (next\_ground\_station\_id, transmission\_rate)]

**elif** transmission\_rate **==** max\_data\_rate:

max\_data\_rate\_path**.**append((ground\_station\_id, transmission\_rate))

max\_data\_rate\_path**.**append((next\_ground\_station\_id, transmission\_rate))

**else**:

max\_data\_rate\_path **=** [(ground\_station\_id, transmission\_rate), (next\_ground\_station\_id, transmission\_rate)]

max\_data\_rate **=** transmission\_rate

routing\_paths**.**append({'Airplane': airplane\_id, 'Routing Path': max\_data\_rate\_path, 'End-to-End Data Rate': max\_data\_rate})

**return** routing\_paths

*# Convert the ground\_stations list to a set to remove duplicates*

ground\_stations **=** list(set(ground\_stations))

*# Call the function to find the routing paths with maximum data transmission rate*

routing\_paths **=** find\_max\_data\_rate\_routing\_paths(airplanes, ground\_stations)

*# Print and store the routing paths in a text file*

**with** open('routing\_paths\_relay.txt', 'w') **as** file:

**for** path **in** routing\_paths:

file**.**write(str(path) **+** '\n\n')

*# Print the routing path and its respective data transmission rates*

*# file.write(f"An example journey is given below (Here is just an example, not a real optimized routing path):\n")*

file**.**write(f"{path['Airplane']}: is the source airplane\n")

**for** i **in** range(len(path['Routing Path'])):

node, rate **=** path['Routing Path'][i]

**if** i **==** 0:

file**.**write(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {path['Airplane']} and {node} is {rate} Mbps.\n")

**else**:

prev\_node, \_ **=** path['Routing Path'][i**-**1]

file**.**write(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {prev\_node} and {node} is {rate} Mbps.\n")

file**.**write(f"End-to-end data rate: '{path['End-to-End Data Rate']}': the final end-to-end data rate is {path['End-to-End Data Rate']} Mbps.\n\n")

*# Print the routing paths and their respective data transmission rates*

**for** path **in** routing\_paths:

print(f"Airplane: {path['Airplane']}")

**for** i **in** range(len(path['Routing Path'])):

node, rate **=** path['Routing Path'][i]

**if** i **==** 0:

print(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {path['Airplane']} and {node} is {rate} Mbps.")

**else**:

prev\_node, \_ **=** path['Routing Path'][i**-**1]

print(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {prev\_node} and {node} is {rate} Mbps.")

print(f"End-to-end data rate: '{path['End-to-End Data Rate']}': the final end-to-end data rate is {path['End-to-End Data Rate']} Mbps.")

print()

**def** find\_max\_data\_rate\_routing\_paths(airplanes, ground\_stations):

routing\_paths **=** []

**for** airplane **in** airplanes:

airplane\_id, x\_airplane, y\_airplane, z\_airplane **=** airplane

max\_data\_rate **=** 0.0

max\_data\_rate\_path **=** []

visited\_ground\_stations **=** set()

**for** i **in** range(len(ground\_stations)):

**for** j **in** range(i **+** 1, len(ground\_stations)):

ground\_station\_id, x\_gs, y\_gs, z\_gs **=** ground\_stations[i]

next\_ground\_station\_id, next\_x\_gs, next\_y\_gs, next\_z\_gs **=** ground\_stations[j]

*# Calculate the distance between the current ground station and the next ground station*

distance **=** calculate\_distance(x\_gs, y\_gs, z\_gs, next\_x\_gs, next\_y\_gs, next\_z\_gs)

*# Calculate the data transmission rate for the link*

transmission\_rate **=** calculate\_transmission\_rate(distance)

**if** transmission\_rate **>** max\_data\_rate:

max\_data\_rate **=** transmission\_rate

max\_data\_rate\_path **=** [(ground\_station\_id, transmission\_rate), (next\_ground\_station\_id, transmission\_rate)]

**elif** transmission\_rate **==** max\_data\_rate:

max\_data\_rate\_path**.**append((ground\_station\_id, transmission\_rate))

max\_data\_rate\_path**.**append((next\_ground\_station\_id, transmission\_rate))

**else**:

max\_data\_rate\_path **=** [(ground\_station\_id, transmission\_rate), (next\_ground\_station\_id, transmission\_rate)]

max\_data\_rate **=** transmission\_rate

visited\_ground\_stations**.**add(ground\_station\_id)

visited\_ground\_stations**.**add(next\_ground\_station\_id)

*# Add remaining ground stations that were not part of the optimal path*

remaining\_ground\_stations **=** [(ground\_station\_id, transmission\_rate) **for** ground\_station\_id, x\_gs, y\_gs, z\_gs **in** ground\_stations **if** ground\_station\_id **not** **in** visited\_ground\_stations]

max\_data\_rate\_path**.**extend(remaining\_ground\_stations)

routing\_paths**.**append({'Airplane': airplane\_id, 'Routing Path': max\_data\_rate\_path, 'End-to-End Data Rate': max\_data\_rate})

**return** routing\_paths

*# Convert the ground\_stations list to a set to remove duplicates*

ground\_stations **=** list(set(ground\_stations))

*# Call the function to find the routing paths with maximum data transmission rate*

routing\_paths **=** find\_max\_data\_rate\_routing\_paths(airplanes, ground\_stations)

*# Print and store the routing paths in a text file*

**with** open('routing\_paths.txt', 'w') **as** file:

**for** path **in** routing\_paths:

file**.**write(str(path) **+** '\n\n')

*# Print the routing paths and their respective data transmission rates*

**for** path **in** routing\_paths:

print(path)

**Multiple objective optimisation**

**def** find\_optimal\_routing\_paths(airplanes, ground\_stations):

routing\_paths **=** []

**for** airplane **in** airplanes:

airplane\_id, x\_airplane, y\_airplane, z\_airplane **=** airplane

optimal\_path **=** []

max\_data\_rate **=** 0.0

min\_latency **=** float('inf')

**for** ground\_station **in** ground\_stations:

ground\_station\_id, x\_gs, y\_gs, z\_gs **=** ground\_station

*# Calculate the distance between the airplane and ground station*

distance **=** calculate\_distance(x\_airplane, y\_airplane, z\_airplane, x\_gs, y\_gs, z\_gs)

*# Calculate the data transmission rate for the link*

transmission\_rate **=** calculate\_transmission\_rate(distance)

**if** transmission\_rate **==** 0:

latency **=** float('inf')

**else**:

*# Calculate the latency for the link*

latency **=** distance **/** transmission\_rate

**if** transmission\_rate **>** max\_data\_rate:

max\_data\_rate **=** transmission\_rate

min\_latency **=** latency

optimal\_path **=** [(ground\_station\_id, max\_data\_rate, min\_latency)]

**elif** transmission\_rate **==** max\_data\_rate **and** latency **<** min\_latency:

min\_latency **=** latency

optimal\_path **=** [(ground\_station\_id, max\_data\_rate, min\_latency)]

routing\_paths**.**append({'Airplane': airplane\_id, 'Optimal Path': optimal\_path, 'End-to-End Data Rate': max\_data\_rate, 'End-to-End Latency': min\_latency})

**return** routing\_paths

*# Call the function to find the optimal routing paths with maximum data transmission rate and minimum latency*

optimal\_routing\_paths **=** find\_optimal\_routing\_paths(airplanes, ground\_stations)

*# Print and store the optimal routing paths in a text file*

**with** open('optimal\_routing\_paths\_length.txt', 'w') **as** file:

**for** path **in** optimal\_routing\_paths:

file**.**write(str(path) **+** '\n')

*# print(path)*

**for** path **in** optimal\_routing\_paths:

print(path)