Flights Graph Analysis

February 23, 2024

1 Flight Connections and Airports Graph Analysis

[ ]:
```
'''
In this analysis, I am going to use graph analysis applied to a Big Data␣
↪framework.
I will use GraphFrames Spark library with the API for DataFrames (GraphFrames␣
↪Spark library)
to analyze a dataset containing information about flight connections and␣
↪airports in the whole world.

I am going to use datasets containing informations about airports, airlines and␣
↪flights worldwide.
Consider these three `csv` files:
- airports.csv: contains one line for each airport in the world. Among the␣
↪others, it provides the columns:
id, name, city, country, iata, latitude and longitude.
- airlines.csv: provides some information for each airline. Among the others,␣
↪it provides the columns:
airline_id, name, country, icao.
- routes.csv enumerates the flights provided by each airline between two␣
↪airports. it provides the columns:
airline_id, airport_source_id, airport_destination_id.

For this Analysis, I am going to use GraphFrames Spark library with the API for␣
↪DataFrames
(GraphFrames Spark library)
'''
```

[ ]:
```
# Input Datasets from the big data cluster
```

[2]:
```
airports = '/data/students/bigdata_internet/lab5/airports.csv'
airlines = '/data/students/bigdata_internet/lab5/airlines.csv'
flights = '/data/students/bigdata_internet/lab5/routes.csv'
```

[ ]:
```
'''In this analysis, PySpark was utilized for its robust distributed computing␣
↪capabilities,
```

```
ideal for handling large datasets efficiently.
If you're using the PySpark shell, no additional setup is necessary.
However, for those working in a Python environment, setting up PySpark involves␣
 ↪the following steps:
1. Install PySpark: Begin by installing PySpark using pip:
pip install pyspark
2. Configure PySpark.sql: In your Python script or interactive session, include␣
 ↪the following configuration
to initialize PySpark.sql:
```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
```
Ensure to execute this configuration before performing any PySpark operations.
For comprehensive installation and configuration instructions, refer to the␣
 ↪official PySpark documentation:
PySpark Installation Guide
'''
```

[ ]:
```
# Reading datasets
```

[3]:
```
airportsDF = spark.read.load(airports, format="csv", header=True,␣
 ↪inferSchema=True, sep=',')
airlinesDF = spark.read.load(airlines, format="csv", header=True,␣
 ↪inferSchema=True, sep=',')
flightsDF = spark.read.load(flights, format="csv", header=True,␣
 ↪inferSchema=True, sep=',')
```

[ ]:
```
# Find top airports and airlines
```

[ ]:
```
# Find countries with more than 200 airports
```

[4]:
```
numberOfAirports = airportsDF.groupBy("country").agg({"id": "count"})
airportsDF.createOrReplaceTempView("airportsDF")
```

[6]:
```
numberOfAirportsSorted = spark.sql('''SELECT country, count(iata) as␣
 ↪number_of_airports FROM airportsDF
                                        GROUP BY country Having count(iata) >␣
 ↪200 ORDER BY number_of_airports DESC''')
```

[7]:
```
numberOfAirportsSorted.show()
```

```
[Stage 7:==================================================>     (179 + 3) / 200]

+-------------+-----------------+
```

```
|      country|number_of_airports|
+-------------+------------------+
|United States|              1512|
|       Canada|               430|
|    Australia|               334|
|       Russia|               264|
|       Brazil|               264|
|      Germany|               249|
|        China|               241|
|       France|               217|
+-------------+------------------+
```

[ ]:
```python
# Find the top-10 airlines by total number of flights
```

[5]:
```python
numberOfFlights = flightsDF.groupBy("airline_id").agg({"airline_id": "count"})
numberOfFlights = numberOfFlights.withColumnRenamed("count(airline_id)",␣
 ↪"num_flights")
joinedAirlineFlights = airlinesDF.join(numberOfFlights, airlinesDF.airline_id␣
 ↪== numberOfFlights.airline_id)
icaoAirline = joinedAirlineFlights.selectExpr("name", "icao", "num_flights")
icaoAirlineOrdered = icaoAirline.sort("num_flights", ascending=False)
icaoAirlineOrderedFinal = icaoAirlineOrdered.limit(10).sort("num_flights",␣
 ↪ascending=True)
```

[153]:
```python
icaoAirlineOrderedFinal.show(truncate=False)
```

```
[Stage 351:====================================================>(199 + 1) / 200]

+-----------------------+----+-----------+
|name                   |icao|num_flights|
+-----------------------+----+-----------+
|easyJet                |EZY |1130       |
|Southwest Airlines     |SWA |1146       |
|Air China              |CCA |1260       |
|China Eastern Airlines |CES |1263       |
|China Southern Airlines|CSN |1454       |
|US Airways             |USA |1960       |
|Delta Air Lines        |DAL |1981       |
|United Airlines        |UAL |2180       |
|American Airlines      |AAL |2354       |
|Ryanair                |RYR |2484       |
+-----------------------+----+-----------+
```

[ ]:
```
# Find the top-5 routes in the world
```

[6]:
```
groupedFlights = flightsDF.groupBy("airport_source_id",␣
↪"airport_destination_id").agg({"airport_source_id": "count"})
```

[7]:
```
groupedFlightsOrdere = groupedFlights.sort("count(airport_source_id)",␣
↪ascending=False)
```

[156]:
```
groupedFlightsOrdere.count()
```

[156]: 37505

[8]:
```
from pyspark.sql.functions import col
joined1 = groupedFlightsOrdere.join(airportsDF, groupedFlightsOrdere.
↪airport_source_id == airportsDF.id).select(col("airport_source_id"),
       col("airport_destination_id"), col("count(airport_source_id)").
↪alias("route_count"), col("name").alias("source_airport"),
       col("city").alias("source_city"))
```

[9]:
```
joined2 = joined1.join(airportsDF, joined1.airport_destination_id == airportsDF.
↪id).select(
    col("source_airport"), col("source_city"), col("name").
↪alias("destination_airport"), col("city").alias("destination_city"),␣
↪col("route_count"))
```

[10]:
```
routes = joined1.join(airportsDF, joined1.airport_destination_id == airportsDF.
↪id).select(col("airport_source_id"), col("airport_destination_id"))
```

[11]:
```
routesName = joined2.sort("route_count", ascending=False)
```

[41]:
```
routesName.show(5, truncate=False)
```

```
[Stage 53:====================================================>      (8 + 1) / 9]

+--------------------------------------------------+-----------+------------------
-------------------------------+-----------------+-----------+
|source_airport
|source_city|destination_airport
|destination_city|route_count|
+--------------------------------------------------+-----------+------------------
-------------------------------+-----------------+-----------+
|Chicago O'Hare International Airport            |Chicago    |Hartsfield Jackson
Atlanta International Airport |Atlanta          |20         |
|Hartsfield Jackson Atlanta International Airport|Atlanta    |Chicago O'Hare
International Airport              |Chicago          |19         |
|Phuket International Airport                    |Phuket     |Suvarnabhumi
```

```
Airport                                    |Bangkok          |13           |
|Chicago O'Hare International Airport             |Chicago     |Louis Armstrong
New Orleans International Airport|New Orleans      |13           |
|Hartsfield Jackson Atlanta International Airport|Atlanta     |Miami
International Airport                             |Miami            |12           |
+-------------------------------------------------+-----------+------------------
-------------------------------+----------------+-----------+
only showing top 5 rows
```

[ ]:
```
# Create the graph of flight connections
# Build a graph using GraphFrames where vertices are the airports in airports.
↪csv,
# and edges are the flights from one airport to another contained in routes.csv
```

[ ]:
```
# Cleaning the flights (routes) by removing missing values
```

[142]:
```
flightsDF.count()
```

[142]: 67663

[10]:
```
flightsClean = flightsDF.filter((flightsDF["airport_source_id"] != "\\N") &␣
↪(flightsDF["airport_destination_id"] != "\\N"))
```

[18]:
```
flightsClean.count()
```

[18]: 67240

[ ]:
```
from graphframes import GraphFrame
```

[ ]:
```
v = airportsDF.select("id", "name", "icao", "city", "country")
v = v.withColumn("id", v.id.cast("string"))
```

[ ]:
```
e = flightsClean.selectExpr("airport_source_id as src", "airport_destination_id␣
↪as dst", "airline_id")
e = e.withColumn("src", e.src.cast("string"))
e = e.withColumn("dst", e.dst.cast("string"))
```

[ ]:
```
g = GraphFrame(v, e)
```

[ ]:
```
# Find the 5 airports with the highest ratio of outgoing edges over incoming␣
↪edges (edgeOut/edgeIn)
```

[ ]:
```
outgoing = g.outDegrees
incoming = g.inDegrees
```

[ ]:
```
outgoing_incoming = outgoing.join(incoming, outgoing.id == incoming.id).select(
    outgoing["id"], incoming["inDegree"], outgoing["outDegree"],␣
↪(outgoing["outDegree"] / incoming["inDegree"]).alias("ratio"))
```

[ ]:
```
outgoing_incoming_final = outgoing_incoming.join(airportsDF, outgoing_incoming.
↪id == airportsDF.id).select(
    airportsDF["name"], outgoing_incoming["id"], outgoing_incoming["inDegree"],␣
↪outgoing_incoming["outDegree"], outgoing_incoming["ratio"])
```

[156]:
```
outgoing_incoming_final.sort("ratio", ascending=False).show(5, truncate=False)
```

```
[Stage 493:============================================>         (163 + 1) / 200]

+---------------------------------------------+----+--------+---------+-----+
|name                                         |id  |inDegree|outDegree|ratio|
+---------------------------------------------+----+--------+---------+-----+
|Bunia Airport                                |1033|1       |3        |3.0  |
|Pikangikum Airport                           |5521|2       |5        |2.5  |
|Transilvania Târgu Mureş International Airport|1662|2       |5        |2.5  |
|Bahir Dar Airport                            |1111|1       |2        |2.0  |
|Ivalo Airport                                |428 |1       |2        |2.0  |
+---------------------------------------------+----+--------+---------+-----+
only showing top 5 rows
```

[ ]:
```
# Finding the number of airports that from there we can reach the city of␣
↪"Torino" taking exactly 1 flight
```

[ ]:
```
# Finding the id of 'Turin Airport'
```

[27]:
```
idTurin = outgoing_incoming_final.filter("name = 'Turin Airport'")
```

[8]:
```
idTurin.show(1)
```

[ ]:
```
# Create the motif
```

[159]:
```
motifs = g.find("(a)-[]->(b)")
```

[ ]:
```
# filter all destination with 'Turin Airport' id
```

[160]:
```
airportsToTurin = motifs.filter("b.id = 1526")
```

[162]:
```
airportsToTurin.select("a").distinct().count()
```

[162]: 29

[ ]:
```
# Find the number of airports that can be reached from the city of "Torino"␣
↪taking exactly 1 flight
```

[163]:
```
airportsFromTurin = motifs.filter("a.id = 1526")
```

[165]:
```
airportsFromTurin.select("b").distinct().count()
```

[165]: 29

[ ]:
```
'''
In the analysis of Turin Airport's network connectivity, both inDegrees and␣
↪outDegrees are observed to be 29.
This symmetry implies a balanced exchange of flights - for each departing␣
↪flight, there is a corresponding
incoming flight. The equal values of inDegrees and outDegrees indicate a␣
↪reciprocity in the airport's connectivity,
highlighting a comprehensive and evenly distributed network.
'''
```

[ ]:
```
# Find the number of airports that can be reached from the city of "Torino"␣
↪taking exactly 2 flights
```

[ ]:
```
# Create the motif
```

[166]:
```
stopFlightMotifs = g.find("(v1)-[]->(v2);(v2)-[]->(v3)")
```

[ ]:
```
# Filter the origin airport 'Turin Airport' id
```

[167]:
```
stopFlightFromTurin = stopFlightMotifs.filter("v1.id = 1526")
```

[168]:
```
stopFlightFromTurin.select("v3").distinct().count()
```

[168]: 590

[ ]:
```
# Find the number of airports that from there we can reach "Los Angeles␣
↪International Airport" using less hop
# than to reach the city of "Torino"
```

[ ]:
```
# Find the id of "Los Angeles International Airport"
```

[21]:
```
idLA = outgoing_incoming_final.filter("name = 'Los Angeles International␣
↪Airport'")
```

[9]:
```
idLA.show(1)
```

[ ]:
```
# Calculate shortestPaths to "Los Angeles International Airport"
```

[24]:
```
toLA = g.shortestPaths(landmarks=["3484"])
```

[ ]:
```
# Calculate shortestPaths to "Turin Airport"
```

[36]:
```
toTurin = g.shortestPaths(landmarks=["1526"])
```

[ ]:
```
# Join two previous dataframes
```

[38]:
```
LA_TO = toLA.join(toTurin, toLA.id == toTurin.id).select(toLA["id"],␣
↪toLA["name"].alias("toLA_name"),
        toLA["distances"].alias("toLA_distance"), toTurin["name"].
↪alias("toTurin_name"), toTurin["distances"].alias("toTurin_distances"))
```

[63]:
```
LA_TO = LA_TO.withColumn("toLA_distance_numeric", col("toLA_distance").
↪getItem("3484")).withColumn(
    "toTurin_distance_numeric", col("toTurin_distances").getItem("1526"))
```

[ ]:
```
# Filter those airports with less hops than Torino
```

[64]:
```
LA_TO_filtered = LA_TO.filter("toLA_distance_numeric <␣
↪toTurin_distance_numeric")
```

[65]:
```
LA_TO_filtered.count()
```

[65]: 1831

```
[ ]: # Find the number of airports that from there we can reach the city of Torino␣
     ↪using less hops than to reach
     # "Los Angeles International Airport"
```

```
[66]: TO_LA_filtered = LA_TO.filter("toLA_distance_numeric >␣
      ↪toTurin_distance_numeric")
```

```
[67]: TO_LA_filtered.count()
```

```
[67]: 94
```

```
[ ]: # Find the number of airports that from there we can reach with the same number␣
     ↪of hops Torino
     # and "Los Angeles International Airport"
```

```
[68]: TOLA_equal_filtered = LA_TO.filter("toLA_distance_numeric ==␣
      ↪toTurin_distance_numeric")
```

```
[69]: TOLA_equal_filtered.count()
```

```
[69]: 1244
```

```
[ ]: # Find the number of connected components of at least two airports are there in␣
     ↪the graph
     # and the size of those connected components.
```

```
[28]: sc.setCheckpointDir("tmp_ckpts")
```

```
23/12/26 09:57:03 WARN spark.SparkContext: Spark is not running in local mode,
therefore the checkpoint directory must not be on the local filesystem.
Directory 'tmp_ckpts' appears to be on the local filesystem.
```

```
[30]: gClean = g.dropIsolatedVertices()
      connComp = gClean.connectedComponents()
      connComp.createOrReplaceTempView("connComp")
      connCompClean = spark.sql('''SELECT component, count(*) FROM connComp GROUP BY␣
      ↪component Having count(*) >= 2''')
```

```
[34]: connCompClean.distinct().count()
```

[34]: 7

[41]:
```
connCompClean.show()
```

```
+-------------+--------+
|    component|count(1)|
+-------------+--------+
|            0|    3188|
|            9|       4|
| 721554505735|       2|
| 266287972363|       4|
| 300647710722|      10|
| 352187318278|       4|
|1460288880648|       2|
+-------------+--------+
```

[ ]:
```
# By considering only the subgraph of the flights that are performed by two␣
↪different airlines (identified by the
# icao), each involving at least 5 citie, I am going to select two airlines to␣
↪show their graphs.
```

[ ]:
```
# Select two Airlines with previous characteristics
# XLA (Excel Airways), GLG (Aerolineas Galapagos)
```

[ ]:
```
test_city = icao_joined.join(airportsDF, icao_joined.src == airportsDF.id).
↪select(icao_joined['icao'], airportsDF['city'])
```

[ ]:
```
test_city.createOrReplaceTempView("test_city")
```

[ ]:
```
select_city = spark.sql('''
                            SELECT icao, COUNT(city)
                            FROM test_city
                            GROUP BY icao
                            HAVING COUNT(city) >= 5''')
```

[ ]:
```
# Create a new dataframe (graphframe) with desired columns
```

[16]:
```
icao_joined = flightsClean.join(airlinesDF, flightsClean.airline_id ==␣
↪airlinesDF.airline_id).select(
            flightsClean["airport_source_id"].alias("src"),␣
↪flightsClean["airport_destination_id"].alias("dst"),
            airlinesDF["airline_id"], airlinesDF["icao"].alias("icao"),␣
↪airlinesDF["name"].alias("airline_name"))
```

```
[ ]: # Create the subgraph of selected airlines
```

```
[17]: icao_joined.createOrReplaceTempView("icao_joined")
e_icao = spark.sql('''SELECT src, dst, icao, airline_name FROM icao_joined␣
 ↪WHERE icao = "XLA" OR icao = "GLG"''')
v_cities = airportsDF.select("id", "city")
g_icao = GraphFrame(v_cities, e_icao)
g_final = g_icao.dropIsolatedVertices()
```

```
[ ]: # Plot the subgraph of these flights
```

```
[64]: from graphviz import Digraph
def vizGraph(edge_list,node_list):
    Gplot=Digraph()
    edges=edge_list.collect()
    nodes=node_list.collect()
    for row in edges:
        Gplot.edge(str(row['src']),str(row['dst']),label=str(row['icao']))
    for row in nodes:
        Gplot.node(str(row['id']),label=str(row['city']))
    return Gplot
```

```
[1]: Gplot=vizGraph(g_final.edges, g_final.vertices)
Gplot
# the picture of final graph will be added at the end of the report.
```

```
[ ]: '''Find the destination airport at minimum distance from "Tancredo Neves␣
 ↪International Airport"
that we can reach by taking exactly 2 flights.I am going to return the␣
 ↪destination airport and its distance
from "Tancredo Neves International Airport" by considering that we cannot come␣
 ↪back to the
Tancredo Neves International Airport'''
```

```
[ ]: # Find the id of "Tancredo Neves International Airport"
```

```
[76]: idTN = outgoing_incoming_final.filter("name = 'Tancredo Neves International␣
 ↪Airport'")
```

```
[2]: idTN.show(1)
```

```
[ ]: # Create the motif that we cannot comeback
```

```
[69]: motifTN = g.find("(a)-[]->(b);(b)-[]->(c); !(c)-[]->(a);!(b)-[]->(a)")
```

[70]:
```
fromTN = motifTN.filter("a.id = 2537")
```

[163]:
```
fromTN.count()
```

```
]]
```

[163]: 39

[71]:
```
destinationList = fromTN.select("c.id")
```

[25]:
```
lat_lon_TN = airportsDF.filter("name = 'Tancredo Neves International Airport'").
↪show(5)
```

[78]:
```
lat_lon = destinationList.join(airportsDF, destinationList.id == airportsDF.id).
↪select(
                    destinationList['id'].alias('id'), airportsDF['name'],␣
↪airportsDF['latitude'], airportsDF['longitude'])
```

[ ]:
```
# Define a function (haversine) to calculate the distance between airports
```

[116]:
```
import math
def haversine(lat1, lon1, lat, lon):
    R = 6371.0
    lat1, lon1, lat, lon = map(math.radians, [lat1, lon1, lat, lon])
    dlat = lat - lat1
    dlon = lon - lon1
    hav = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat) * math.
↪sin(dlon / 2) ** 2
    distance = 2 * R * math.asin(math.sqrt(hav))
    return distance
```

[ ]:
```
# Register the function
```

[105]:
```
spark.udf.register('hav', haversine)
```

```
24/01/09 17:57:13 WARN analysis.SimpleFunctionRegistry: The function hav
replaced a previously registered function.
```

[105]:
```
<function __main__.haversine(lat1, lon1, lat, lon)>
```

[106]:
```
dinstanceAirportsDF = lat_lon.selectExpr("id", "name", "hav(-19.62444305419922,␣
↪-43.97194290161133, latitude, longitude) as distance").sort("distance")
```

[118]:
```
dinstanceAirportsDF.show(1, truncate = False)
```

```
[Stage 596:=====================================================>(198 + 2) / 200]
```

```
+----+----------------------------------+------------------+
|id  |name                              |distance          |
+----+----------------------------------+------------------+
|2555|Hercílio Luz International Airport|1008.7177113996354|
+----+----------------------------------+------------------+
only showing top 1 row
```

[ ]:
```
'''There are 39 airports  that can be reach by taking exactly two flights from
"Tancredo Neves International Airport" that there is no return flight from␣
↪those destination.
The minimum distance from this airport with these conditions is 1008.
↪7177113996354 which the destination
airport is "Hercílio Luz International Airport".'''
```

[ ]:
```
# Compute the total flown distance in kilometers, considering the distance from
# "Tancredo Neves International Airport" to the first airport summed to the␣
↪distance from the first airport to
# second one (total flown distance)
```

[94]:
```
bothAirportsList = fromTN.select("b.id", "c.id").filter("c.id = 2555")
```

[26]:
```
bothAirportsList.show()
```

[98]:
```
airportB_List = fromTN.select("b.id")
```

[101]:
```
lat_lon_B = airportB_List.join(airportsDF, airportB_List.id == airportsDF.id).
↪select(
                    airportB_List['id'].alias('id'), airportsDF['name'],␣
↪airportsDF['latitude'], airportsDF['longitude'])
```

[27]:
```
lat_lon_B.filter("id = 2442").show(1, truncate=False)
```

[121]:
```
airport_B_C = lat_lon.selectExpr("id", "name", "hav(-34.5592, -58.4156,␣
↪latitude, longitude) as distance").filter("id = 2555")
```

[122]:
```
airport_B_C.show(truncate=False)
```

```
[Stage 686:==========================================>              (58 + 17) / 75]

+----+--------------------+------------------+
|  id|                name|          distance|
+----+--------------------+------------------+
|2555|Hercílio Luz Inte…|1210.6138281258875|
```

```
+----+--------------------+------------------+
```

```
[119]: airport_A_B = lat_lon_B.selectExpr("id", "name", "hav(-19.62444305419922, -43.
        ↪97194290161133, latitude, longitude) as distance")
```

```
[120]: airport_A_B.show(1, truncate=False)
```

```
[Stage 632:================================================>        (7 + 1) / 8]

+----+---------------------+------------------+
|id  |name                 |distance          |
+----+---------------------+------------------+
|2442|Jorge Newbery Airpark|2186.1514575333313|
+----+---------------------+------------------+
only showing top 1 row
```

```
[ ]: '''The distance between "Tancredo Neves International Airport" and "Jorge␣
      ↪Newbery Airpark" is 2186.1514575333313
     and the distance between "Jorge Newbery Airpark" and "Hercílio Luz␣
      ↪International Airport" is 1210.6138281258875,
     so the sum of distances is 3396.76529.'''
```