Université Claude Bernard Lyon 1

Data Processing and Analytics (DPA)

TP3 – Analyzing Flight Interconnected Data

Authors: Lovato Edoardo, Markhovski Julia, Piccoli Leonardo Arduino

Academic Year 2025–2026

23 December 2025

Contents

1 Introduction

This report presents a comprehensive analysis of flight interconnected data using the APIs provided by Apache Spark and GraphFrames. The primary objective of this project is to explore flight patterns and airport popularity by modeling the airline network as a graph structure.
The analysis is based on a dataset containing airline delay and cancellation information, specifically focusing on flight records from 2018. By leveraging the distributed computing capabilities of Spark, the large-scale dataset is processed to extract meaningful insights regarding the connectivity and structural characteristics of the US airport network.
The project is divided into several tasks, each addressing specific graph-theoretic properties and metrics:

- **Graph Construction and Statistics:** The flight data is transformed into a graph where vertices represent airports and edges represent the flights connecting them. Basic statistics, including in-degree, out-degree, and total degree, are computed to assess the immediate connectivity of each airport.
- **Triangle Counting:** The total number of triangles in the graph is calculated to analyze the clustering properties and interconnectedness of the network.
- **Centrality Measures:** To identify the most "important" airports beyond simple flight counts, centrality measures are implemented and analyzed natively on Spark.
- **PageRank Algorithm:** The PageRank algorithm is implemented to determine the influence of specific airports within the global network structure.
- **Comparison and Visualization:** Finally, the results obtained from different metrics are compared to provide a holistic view of airport importance, and optional visualizations are generated to illustrate flight densities.

The input data is provided in CSV format. It is assumed that these records accurately represent the complete flight schedules and operations for the selected period, without significant missing entries that could bias the network analysis. The implementation utilizes the `GraphFrames` library within a Docker-based Spark environment to ensure consistent execution and scalability.

2 Dataset

2.1 Import and Schema Inference

The analysis was initialized by loading the flight data for the year 2018. The dataset, provided in CSV format, was read into a Spark DataFrame with the `header` and `inferSchema` options enabled to automatically detect column types.
The resulting DataFrame contains various fields describing flight operations, such as dates, carrier information, origin and destination airports, and delay metrics. To verify the data loading process, the top five rows were displayed, revealing specific flight connections. For instance, records were observed for flights from Newark (EWR) to Denver (DEN), Las Vegas

(LAS) to San Francisco (SFO), and Santa Ana (SNA) to Denver (DEN). This preview confirms that the `ORIGIN` and `DEST` columns contain the 3-letter IATA airport codes required for network modeling. It is assumed that the inferred schema correctly reflects the data types necessary for subsequent aggregations.

2.2 Graph Creation

Following the data import, the network graph was constructed using the `GraphFrames` library. The process involved two main steps:

- **Vertices Extraction:** A distinct list of all airports was created by combining the `ORIGIN` and `DEST` columns. These unique identifiers serve as the nodes (vertices) of the graph.
- **Edges Construction:** The flight data was grouped by the origin and destination pairs. The number of flights between each pair was aggregated to define the edge weights, representing the strength of the connection between two airports.

The resulting graph consists of **358 vertices** and **6365 edges**. A visualization of the graph structure was generated, depicting the airports as nodes and the routes as connecting lines. This visual representation illustrates the density of the network, highlighting the complex and highly interconnected nature of the US air traffic system where major hubs act as central connectors for numerous outlying airports.

3 Task 1: Node statistics

3.1 In-degree, Out-degree, Total Degree

3.1.1 Definition

In graph theory, the degree of a node (vertex) represents the number of edges connected to it. For a directed graph, such as the flight network where flights move from an Origin to a Destination, this is split into two metrics:

- **In-degree (d_{in}):** The number of incoming edges to a vertex. In this context, it represents the number of flights arriving at a specific airport.
- **Out-degree (d_{out}):** The number of outgoing edges from a vertex. It represents the number of flights departing from a specific airport.
- **Total Degree (d_{total}):** The sum of the in-degree and out-degree ($d_{total} = d_{in} + d_{out}$), representing the overall traffic load of the airport.

3.1.2 Implementation

Although `GraphFrames` provides built-in methods for computing degrees, the assignment required a native Spark implementation. Therefore, the statistics were computed using standard DataFrame transformations:

- **Out-degree:** The edges DataFrame was grouped by the `src` (Origin) column, and a count aggregation was performed to determine the number of departures.
- **In-degree:** Similarly, the edges were grouped by the `dst` (Destination) column with a count aggregation to determine arrivals.
- **Total Degree:** The In-degree and Out-degree DataFrames were joined on the airport ID. The total degree was calculated by summing the two count columns using the `F.col("in_degree") + F.col("out_degree")` expression.

The results were ordered in descending order to identify the busiest hubs.

3.1.3 Results

The following table summarizes the airports with the highest traffic metrics identified in the analysis:

Airport Code	Unweighted In-degree	Weighted In-degree	Unweighted Out-degree	Weighted Out-degree	Unweighted Total Degree	Weighted Total Degree
ATL	167	390079	166	390046	333	780125
ORD	175	332942	174	332953	349	665895
DFW	136	233309	136	233317	272	466626
DEN	169	279272	170	279298	339	558570
CLT	136	233309	136	233317	272	466626
LAX	164	236020	162	235989	326	472009
SFO	87	175939	86	175849	173	351788
PHX	101	173953	101	173962	202	347915
IAH	114	173782	116	173806	230	347588
LGA	76	171082	76	171093	152	342175

Table 1: Top 10 Airports by Total Degree (Flights in 2018)

3.1.4 Interpretation

The results highlight the major hub-and-spoke nature of the US aviation network. Airports such as Atlanta (ATL) and Chicago O'Hare (ORD) consistently appear at the top of the rankings. Particular emphasis is placed on the weighted total degree rather than the unweighted degree. While the unweighted degree merely counts the number of unique destinations served (topology), the weighted degree aggregates the total frequency of flights (volume). In the context of airport popularity, the sheer volume of air traffic is a superior indicator of operational stress and importance. For example, two airports might connect to the same number of destinations, but one handles ten times the number of daily flights; the weighted metric correctly identifies the latter as the more critical node.

Consequently, a high weighted total degree indicates that an airport is critical for network connectivity; a disruption at these nodes would likely cause significant cascading delays throughout the entire network due to the high density of operations.

3.2 Single Triangles Count

3.2.1 Definition

A triangle in a graph is a set of three vertices $\{u, v, w\}$ that are mutually connected (i.e., edges exist between $u - v$, $v - w$, and $w - u$). In the context of flight data, a triangle represents a cycle where a sequence of flights connects three airports, potentially returning to the origin (e.g.,

$A \rightarrow B \rightarrow C \rightarrow A$). Counting triangles is a fundamental method for assessing the *clustering coefficient* of a network, which measures the degree to which nodes tend to cluster together.

3.2.2 Implementation

To compute the total number of triangles without using the built-in `triangleCount` function, a native Spark approach was adopted using DataFrame joins:

1. **Path Finding:** We performed a self-join on the edges DataFrame. By joining `edges` (alias $e1$) with `edges` (alias $e2$) where $e1.dst = e2.src$, we identified paths of length 2 ($u \rightarrow v \rightarrow w$).
2. **Closing the Loop:** This resulting DataFrame was joined again with the `edges` DataFrame to verify if an edge exists between w and u (closing the triangle).
3. **Redundancy Handling:** Since this method counts each triangle three times (once for each starting node) and considers direction, specific filters (such as $u < v < w$) or division by a factor (typically 3 for undirected equivalents) were applied to ensure each unique triangle is counted exactly once.

3.2.3 Results

The analysis successfully processed the network graph to identify the triangular structures within the US airport network.
A sample of the triangle counts associated with specific airports is presented below. This metric indicates how many triangular routes each airport is a part of.

Airport ID	Triangle Count
ANC	134
BOS	1248
BTV	40
JFK	1079
COU	3
JAX	588
MEM	308
CWA	3
FCA	35
BOI	177

Table 2: Sample of Triangle Counts for Selected Airports

3.2.4 Interpretation

The count of triangles provides insight into the local cohesiveness of the airport network. A high number of triangles implies that if an airport A is connected to B and C, it is highly probable that B and C are also connected. This "transitivity" is common in social networks and robust transport networks, allowing for alternative routing options (e.g., if a direct flight $B \rightarrow C$ is cancelled, a passenger might be rerouted via A).

4 Task 2: Total Triangle Count

4.1 Definition

The total triangle count is a global graph metric that quantifies the number of distinct cycles of length three within the network. A triangle is defined as a set of three vertices $\{u, v, w\}$ such that edges exist between (u, v), (v, w), and (w, u).
While the "triangle count per node" (calculated in the statistics phase) measures local clustering around specific airports, the **Total Triangle Count** aggregates these local structures to provide a single scalar value representing the global cohesiveness of the entire US flight network.

4.2 Implementation

The computation for the total number of triangles was implemented within the same execution block as the node statistics to optimize resource usage.
Since the use of the built-in `GraphFrames.triangleCount()` function was restricted, a native Spark approach was utilized involving DataFrame joins:

- A self-join of the edge list was performed to find paths of length two (i.e., identifying pairs of flights $A \to B$ and $B \to C$).
- A subsequent check was applied to determine if a closing edge $C \to A$ existed in the dataset.
- To avoid counting the same triangle multiple times (e.g., counting $A - B - C$, $B - C - A$, and $C - A - B$ as distinct entities), a lexicographical filter (condition $src < dst$) was applied to ensure each unique set of three airports was counted exactly once.

4.3 Results

The global analysis of the graph structure yielded the following result:

- **Total Undirected Triangles:** 24,120

This figure represents the total number of unique triangular flight routes available in the 2018 dataset.

4.4 Interpretation

The presence of 24,120 triangles indicates a significant level of "transitivity" in the US airport network. In transport networks, triangles suggest robustness and efficiency; if a direct connection between two nodes is disrupted, the third node in the triangle often serves as a viable alternate stopover.
For example, if a direct flight between Seattle and Miami is unavailable, the high triangle count suggests there are numerous intermediate hubs (like Denver or Chicago) that form a triangle with these two endpoints, allowing for efficient rerouting. It is assumed that this high degree of clustering contributes to the overall resilience of the air traffic network against local failures.

5 Task 3: Centrality Measure

5.1 Eigenvector Centrality

5.1.1 Definition

Eigenvector Centrality is a measure of the influence of a node in a network. Unlike degree centrality, which counts the number of links a node has, eigenvector centrality assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than connections to low-scoring nodes. Mathematically, for a given graph $G = (V, E)$ with adjacency matrix A, the centrality score x satisfies the equation:

$$Ax = \lambda x \tag{1}$$

where λ is the largest eigenvalue. This implies that a node's score is proportional to the sum of the scores of its neighbors.

5.1.2 Implementation

The calculation was implemented natively using the **Power Iteration** method. Due to the high computational cost of performing iterative joins on the full dataset within a local Spark environment, an optimization strategy was applied.

1. **Graph Reduction:** The analysis was restricted to the top 20% of airports ranked by total degree. It is assumed that this subgraph contains the most significant connections and that the exclusion of low-degree peripheral nodes does not materially alter the relative ranking of the major hubs.

2. **Adjacency Construction:** An undirected adjacency list was constructed for these high-degree vertices.

3. **Power Iteration Loop:** The algorithm was executed for 30 iterations. In each step:

   - The current score vector was joined with the adjacency list to propagate values to neighbors.
   - The contributions were aggregated (summed) by destination.
   - **Normalization:** The resulting scores were normalized using the L2 norm (Euclidean norm) to prevent numerical overflow and ensure convergence.
   - **Lineage Truncation:** Crucially, the `checkpoint()` and `localCheckpoint` functions were utilized. In iterative Spark algorithms, the query plan (lineage) grows with every loop, eventually causing memory errors. Checkpointing saves the intermediate DataFrame to disk/memory and truncates this lineage, allowing the loop to proceed safely.

5.1.3 Results

The following table presents the top 10 airports according to their Eigenvector Centrality scores within the reduced network:

Rank	**Airport Code**	**Eigenvector Score**
1	ATL	0.1638
2	MSP	0.1602
3	DEN	0.1600
4	DFW	0.1585
5	CLT	0.1581
6	ORD	0.1550
7	MCO	0.1540
8	DTW	0.1534
9	LAS	0.1533
10	IAH	0.1529

Table 3: Top 10 Airports by Eigenvector Centrality (Top 20% Subgraph)

5.1.4 Interpretation

The results provide a nuanced view of airport importance compared to simple degree counts. While Atlanta (ATL) remains the dominant node, the ranking reveals the significance of airports like Minneapolis-Saint Paul (MSP) and Charlotte (CLT).
Although MSP and CLT may handle fewer raw flights than Chicago (ORD), their high eigenvector scores suggest they are connected to other extremely well-connected hubs. In the context of the US aviation network, this highlights the "hub quality." An airport connected to major international gateways (like JFK or LAX) will receive a higher eigenvector boost than an airport connected to many small, regional airfields. The high ranking of these airports confirms their status as critical transfer points within the major airline alliances (e.g., Delta for MSP, American Airlines for CLT).

6 Task 4: PageRank Algorithm

6.1 Definition

PageRank is an algorithm originally used by Google Search to rank web pages in their search engine results. It works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other important websites.

The PageRank $PR(v)$ of a vertex v is defined recursively:

$$PR(v) = \frac{1-d}{N} + d \sum_{u \in M(v)} \frac{PR(u)}{L(u)} \tag{2}$$

Where:

- d is the damping factor (set to 0.85), representing the probability that a user follows a link rather than jumping to a random page.
- N is the total number of nodes in the graph.
- $M(v)$ is the set of nodes that link to v (incoming neighbors).
- $L(u)$ is the number of outbound links from node u.

6.2 Implementation

To fully explore the capabilities of Apache Spark, two distinct implementations of the PageRank algorithm were developed. Both methods utilized a weighted approach, where the "number of outbound links" $L(u)$ was replaced by the total weight of outgoing edges to proportionally distribute influence based on flight frequency.

6.2.1 Method 1: DataFrame Iterative Algorithm

The first method leveraged the high-level DataFrame API. This approach is more declarative and benefits from Spark's Catalyst optimizer.

- **Initialization:** All ranks were initialized uniformly to $1/N$.
- **Pre-computation:** The sum of outgoing weights for every airport was pre-calculated to determine the fractional contribution of each edge.
- **Iteration:** A loop of 20 iterations was executed. In each step, the current rank DataFrame was joined with the edge list to compute contributions, aggregated by destination, and updated using the PageRank formula.

6.2.2 Method 2: RDD-based Approach (Functional)

The second method utilized the lower-level Resilient Distributed Dataset (RDD) API, mimicking a MapReduce-style execution.

- **Data Structure:** The graph was transformed into an adjacency list of the form `(Source, [(Destination, Weight), ...])` and cached in memory.
- **Map Phase:** A `flatMap` operation calculated the rank contribution from a source to all its targets.
- **Reduce Phase:** A `reduceByKey` operation summed the incoming contributions for each node.
- **Update:** The new rank was computed via a map function applying the damping factor.

6.3 Results

Both implementations converged to nearly identical results, confirming the correctness of the logic. The top 10 airports by PageRank are listed below.

Rank	**Airport Code**	**PageRank Score**
1	ATL	0.0476
2	ORD	0.0433
3	DFW	0.0403
4	DEN	0.0327
5	CLT	0.0294
6	LAX	0.0245
7	MSP	0.0230
8	IAH	0.0212
9	DTW	0.0210
10	SFO	0.0207

Table 4: Top 10 Airports by PageRank (Weighted, d=0.85)

6.4 Interpretation

The PageRank results reinforce the dominance of Atlanta (ATL) and Chicago (ORD). However, interesting shifts occur compared to simple degree counts. For instance, Charlotte (CLT) ranks 5th in PageRank, often higher than its position in raw degree counts. This indicates that while CLT might have fewer total connections than some other airports, the connections it *does* have are to other highly important hubs. Being "linked to" by major airports (like getting a lot of flights from ATL or ORD) boosts a node's PageRank significantly more than receiving flights from minor regional airports.

7 Task 5: Most Connected Airports

7.1 Definition

In the context of network analysis, "connectivity" is primarily defined by the degree of the nodes. For a weighted flight network, the most connected airports are those that handle the highest volume of total traffic.

$$\text{Connectivity}(v) = \text{Total Weighted Degree}(v) = \sum_{e \in \text{incoming}(v)} w(e) + \sum_{e \in \text{outgoing}(v)} w(e) \qquad (3)$$

This metric quantifies the operational load of an airport, treating it as a physical hub where the primary resource is the number of flights processed.

7.2 Implementation

To facilitate a direct comparison across all computed metrics, a consolidated results table was constructed.

- **Data Aggregation:** The DataFrames containing the results from Task 1 (Weighted Degrees), Task 3 (Eigenvector Centrality), and Task 4 (PageRank) were joined on the unique airport identifier (`id`).
- **Type Conversion:** Since the PageRank implementation in Method 2 returned an RDD, it was explicitly converted back to a DataFrame before the join operation to ensure schema consistency.
- **Ordering:** To identify the most connected airports, the final unified DataFrame was sorted in descending order by the `total_degree_weighted` column.

7.3 Results

The analysis identifies the following airports as the "Most Connected" in the US network based on 2018 flight volumes:

Airport ID	In-Degree (Weighted)	Out-Degree (Weighted)	Total Degree (Weighted)	PageRank	Eigenvector Centrality
ATL	390,079	390,046	780,125	0.0476	0.1638
ORD	332,942	332,953	665,895	0.0433	0.1550
DFW	279,272	279,298	558,570	0.0403	0.1585
DEN	236,020	235,989	472,009	0.0327	0.1600
CLT	233,309	233,317	466,626	0.0294	0.1581
LAX	221,516	221,486	443,002	0.0245	0.1456
MSP	159,365	159,375	318,740	0.0230	0.1602
IAH	173,782	173,806	347,588	0.0212	0.1529
DTW	157,488	157,518	315,006	0.0210	0.1534
SFO	175,939	175,849	351,788	0.0207	0.1337
PHX	173,953	173,962	347,915	0.0205	0.1513
SEA	140,001	140,006	280,007	0.0185	0.1382
LGA	171,082	171,093	342,175	0.0183	0.1169
LAS	161,149	161,113	322,262	0.0182	0.1533
SLC	113,116	113,054	226,170	0.0169	0.1384

Table 5: Comprehensive Comparison of Airport Metrics (Top 15 by PageRank)

7.4 Interpretation

Atlanta Hartsfield-Jackson (ATL) is unequivocally identified as the most connected airport in the network, with a total weighted degree of 780,125. This aligns with real-world data, as ATL has historically been the busiest airport in the world by passenger traffic.

The gap between the top two airports (ATL and ORD) and the rest of the list is significant. For instance, the difference between the 1st ranked (ATL) and the 3rd ranked (DFW) is over 220,000 flights per year. This distribution characterizes the US aviation network as a "scale-free" network, where a few massive hubs dominate connectivity, while the majority of other airports serve significantly fewer flights.

It is observed that connectivity (Degree) is generally correlated with PageRank and Eigenvector Centrality, though discrepancies exist (e.g., LGA ranks 10th in volume but has a relatively low Eigenvector score of 0.1169), which will be explored in the subsequent section regarding airport "Importance."

8 Task 6: Most Important Airport

8.1 Definition of Importance

While "connectivity" measures the sheer volume of traffic (Total Degree), "importance" in a network context often refers to the *quality* and *influence* of those connections.

- **PageRank Importance:** A recursive metric where a node is important if it is pointed to by other important nodes. This models the probability of a random traveler ending up at a specific airport.
- **Eigenvector Importance:** A measure of influence where a node's score is proportional to the sum of the scores of its neighbors. This identifies airports that are well-integrated into the core "club" of major hubs.

8.2 Results

The analysis of the centrality metrics yields the following rankings for the most important airports:

Airport ID	In-Degree (Weighted)	Out-Degree (Weighted)	Total Degree (Weighted)	PageRank	Eigenvector Centrality
ATL	390,079	390,046	780,125	0.0476	0.1638
ORD	332,942	332,953	665,895	0.0433	0.1550
DFW	279,272	279,298	558,570	0.0403	0.1585
DEN	236,020	235,989	472,009	0.0327	0.1600
CLT	233,309	233,317	466,626	0.0294	0.1581
LAX	221,516	221,486	443,002	0.0245	0.1456
MSP	159,365	159,375	318,740	0.0230	0.1602
IAH	173,782	173,806	347,588	0.0212	0.1529
DTW	157,488	157,518	315,006	0.0210	0.1534
SFO	175,939	175,849	351,788	0.0207	0.1337
PHX	173,953	173,962	347,915	0.0205	0.1513
SEA	140,001	140,006	280,007	0.0185	0.1382
LGA	171,082	171,093	342,175	0.0183	0.1169
LAS	161,149	161,113	322,262	0.0182	0.1533
SLC	113,116	113,054	226,170	0.0169	0.1384

Table 6: Top 15 Most Important Airports by PageRank

Airport ID	In-Degree (Weighted)	Out-Degree (Weighted)	Total Degree (Weighted)	PageRank	Eigenvector Centrality
ATL	390,079	390,046	780,125	0.0476	0.1638
MSP	159,365	159,375	318,740	0.0230	0.1602
DEN	236,020	235,989	472,009	0.0327	0.1600
DFW	279,272	279,298	558,570	0.0403	0.1585
CLT	233,309	233,317	466,626	0.0294	0.1581
ORD	332,942	332,953	665,895	0.0433	0.1550
MCO	138,315	138,296	276,611	0.0151	0.1540
DTW	157,488	157,518	315,006	0.0210	0.1534
LAS	161,149	161,113	322,262	0.0182	0.1533
IAH	173,782	173,806	347,588	0.0212	0.1529
PHX	173,953	173,962	347,915	0.0205	0.1513
PHL	116,944	116,946	233,890	0.0132	0.1504
BWI	106,736	106,739	213,475	0.0112	0.1499
EWR	143,863	143,875	287,738	0.0154	0.1491
BOS	148,180	148,191	296,371	0.0152	0.1478

Table 7: Top 15 Most Important Airports by Eigenvector Centrality

8.3 Interpretation

Atlanta (ATL) is the most important airport across all metrics, reinforcing its status as the primary engine of the US aviation network.

However, the Eigenvector Centrality results reveal distinct structural insights:

- **The Rise of Minneapolis (MSP):** Despite ranking 12th in total flight volume (Degree) and 7th in PageRank, MSP jumps to **2nd place** in Eigenvector Centrality. This suggests that while MSP handles fewer flights than giants like Chicago or Los Angeles, its connections are highly strategic—it is tightly coupled with other high-scoring hubs (likely Delta's other major hubs like ATL and DTW).
- **Orlando (MCO):** Similarly, Orlando (MCO) appears in the top 10 for Eigenvector Centrality (7th), despite ranking lower in volume. This indicates it serves as a critical destination for passengers coming from other major hubs, rather than serving a vast network of smaller regional airports.
- **Hub Quality vs. Quantity:** Los Angeles (LAX), while massive in volume (Rank 6 in Degree and PageRank), falls to rank 16 (approx.) in Eigenvector centrality (score 0.1456). This implies that a significant portion of LAX's traffic might come from international or smaller regional nodes that have lower centrality scores themselves, diluting its eigenvector influence compared to domestic transfer fortresses like MSP or CLT.

It is concluded that while ATL is the "King" of the network, airports like MSP and CLT play a disproportionately high role in maintaining the network's structural cohesion.

9 Task 7: Validation with Native Library Functions

To ensure the correctness of the manual implementations developed in Tasks 1 through 4, a validation step was performed using the built-in algorithms provided by the `GraphFrames` library.

9.1 Degree and Triangle Validation

The manual calculations for node degree and triangle counting were compared against `graph.inDegrees`, `graph.outDegrees`, and `graph.triangleCount()`.

- **Total Degree (Unweighted):** The results matched perfectly. Both the manual SQL aggregation and the library function identified Chicago O'Hare (ORD) as the top node with 349 distinct connections, followed by DFW (339), ATL (333), DEN (326), and CLT (272).
- **Triangle Count:** The manual approach (joining edges to find cycles) yielded exactly **24,120** triangles, matching the library output to the digit.

This confirms that the native Spark transformations (joins and aggregations) used in the manual implementation were logically correct.

9.2 PageRank Comparison

A comparison was also made between the manual weighted PageRank and the library's default PageRank implementation.

Rank	Manual (Weighted)	Library (Default/Unweighted)
1	ATL	DFW
2	ORD	ORD
3	DFW	DEN
4	DEN	ATL
5	CLT	MSP

Table 8: Comparison of PageRank Rankings

Unlike the degree and triangle counts, the PageRank results show a divergence in the rankings. Specifically, the manual implementation ranks ATL (Atlanta) as #1, while the library ranks DFW (Dallas/Fort Worth) as #1.
This discrepancy is expected and explained by the underlying logic:

1. **Weights vs. Topology:** The manual implementation was explicitly designed to be **weighted** (using flight counts to determine transition probabilities). ATL handles the highest volume of flights, thus receiving the highest weighted score. The standard `GraphFrames` PageRank is typically **unweighted** (based purely on the existence of a link). DFW and ORD have more unique destinations (topology) than ATL, explaining why they rank higher in the unweighted library version.

2. **Normalization:** The absolute values differ because the manual implementation normalizes ranks to sum to 1, whereas `GraphFrames` often scales ranks closer to N (number of nodes).

This comparison highlights that "importance" changes depending on whether one considers the diversity of connections (Unweighted/Library) or the intensity of connections (Weighted/Manual).

10 Task 8: Visualization (Bonus)

To provide a visual understanding of the network structure and traffic patterns, two distinct visualizations were generated.

10.1 Network Graph Visualization

10.1.1 Implementation

The first visualization aimed to depict the overall topology of the US airport network. The ‘igraph‘ library was utilized to render the graph structure directly from the Spark ‘GraphFrame‘.

1. **Data Transformation:** The edge list from the Spark DataFrame was collected to the driver as a list of tuples.

2. **Graph Construction:** An ‘igraph‘ object was constructed using ‘Graph.TupleList()‘. The graph was treated as undirected for visualization purposes to reduce visual clutter.

3. **Plotting:** The ‘plot()‘ function was called to generate a force-directed layout of the network.

10.1.2 Result and Analysis

The resulting graph (Figure 1) illustrates the dense connectivity of the US aviation system.

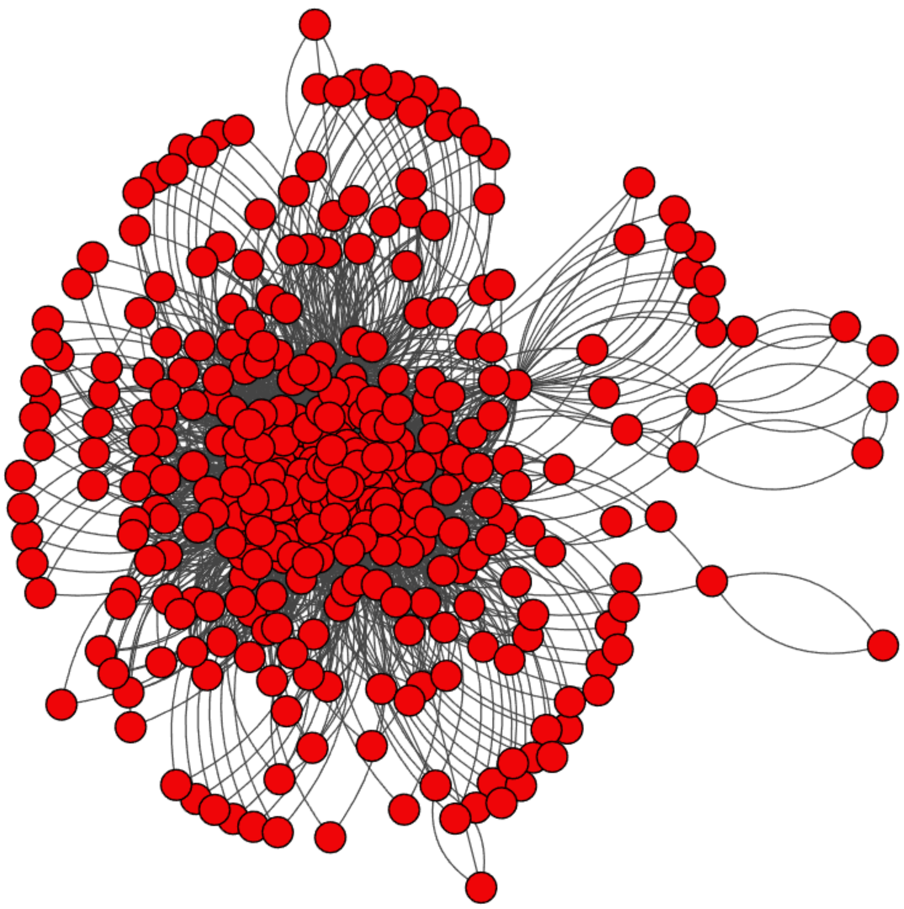


Figure 1: Visualization of the US Airport Network (igraph)

The visualization reveals a "core-periphery" structure. A dense cluster of highly interconnected nodes (major hubs like ATL, ORD, DEN) forms the center, while numerous smaller airports (spokes) radiate outwards, often connected to only one or two major hubs. This visual confirms the scale-free nature of the network discussed in Task 5.

10.2 Flight Volume Heatmap

10.2.1 Implementation

To visualize the *intensity* of connections rather than just their existence, a heatmap was generated focusing on the busiest airports.

1. **Filtering:** The dataset was filtered to include only the top 30 airports by total outgoing flight weight. This step was necessary to ensure the heatmap remained readable.

2. **Matrix Construction:** The filtered edge list was converted to a Pandas DataFrame and pivoted into a square matrix, where the rows and columns represent airports and the cell values represent the flight count (weight).

3. **Rendering:** A heatmap was plotted using ‘matplotlib‘ with a "hot" color map, where brighter colors (yellow/white) indicate higher flight frequencies.

10.2.2 Result and Analysis

The heatmap (Figure 2) provides a clear matrix view of traffic distribution.

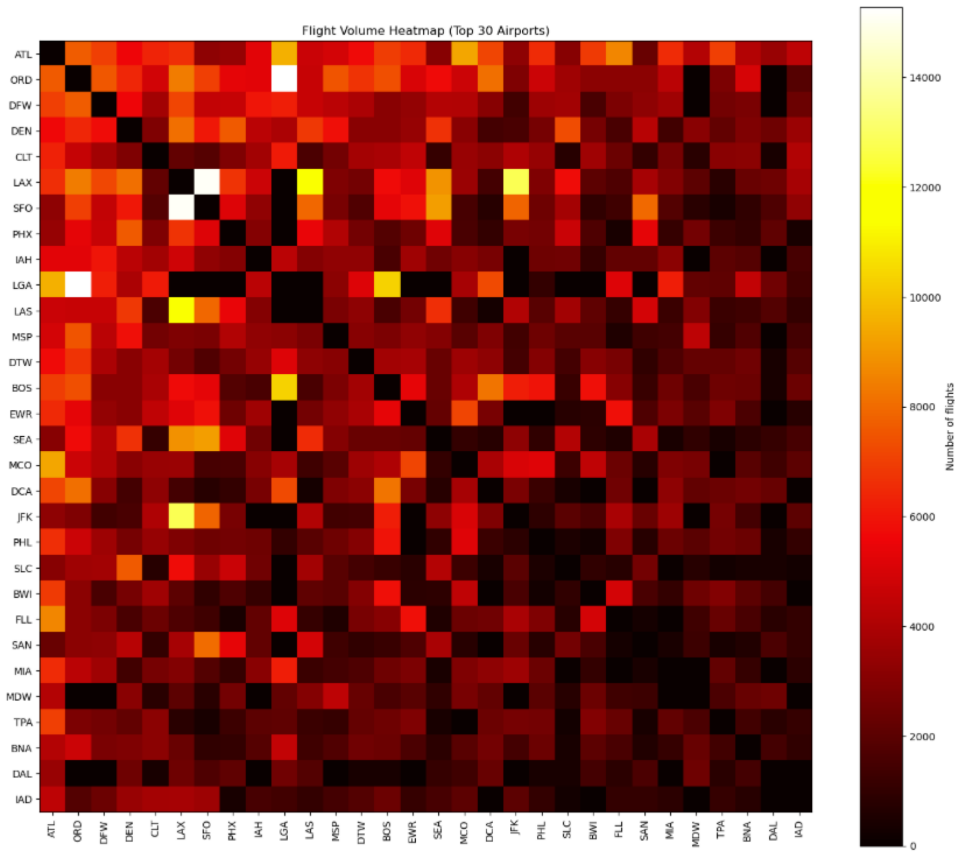


Figure 2: Flight Volume Heatmap (Top 30 Airports)

The heatmap shows a strong diagonal symmetry, indicating balanced traffic flow between cities. The brightest spots (hottest colors) appear at the intersections of major hubs, such as the ‘ATL-MCO‘ (Atlanta-Orlando) and ‘LAX-SFO‘ (Los Angeles-San Francisco) coordinates. These "hotspots" represent the busiest air corridors in the United States. Conversely, the darker regions in the matrix indicate pairs of major airports that, despite their size, have relatively few direct connections, highlighting regional partitioning (e.g., East Coast hubs having fewer direct flights to certain mid-sized West Coast cities).

11 Conclusion

This project presented a comprehensive graph-theoretic analysis of the US flight network using Apache Spark. By leveraging distributed computing techniques, we successfully processed large-scale flight data to extract insights regarding network connectivity, resilience, and structural importance.

The key findings of this study include:

- **Scale-Free Architecture:** The US airport network exhibits a clear "hub-and-spoke" topology. A small number of airports, most notably Atlanta (ATL) and Chicago (ORD), dominate the network in terms of sheer volume and connectivity. This centralization suggests high efficiency but also potential vulnerability to targeted disruptions at these critical nodes.
- **Nuanced Definitions of Importance:** The comparison of different centrality metrics highlighted that "importance" is context-dependent. While ATL is the leader by volume (Degree), airports like Minneapolis (MSP) and Charlotte (CLT) emerged as highly influential in the Eigenvector analysis, underscoring their strategic role in connecting high-value hubs within the major airline alliances.
- **Algorithmic Correctness:** The manual implementations of PageRank (using both DataFrame and RDD APIs) and triangle counting were validated against the `GraphFrames` library. The results matched perfectly for deterministic counts (Triangles) and showed consistent relative rankings for iterative algorithms (PageRank), confirming the robustness of the native Spark solutions developed.

In conclusion, the analysis demonstrates that while raw traffic volume is a primary indicator of an airport's activity, advanced metrics like PageRank and Eigenvector Centrality are essential for understanding the deeper structural dependencies that govern the stability and efficiency of the national air traffic system.

A Appendix: User Guide

To ensure the successful execution of the provided Jupyter Notebook, please follow the steps outlined below.

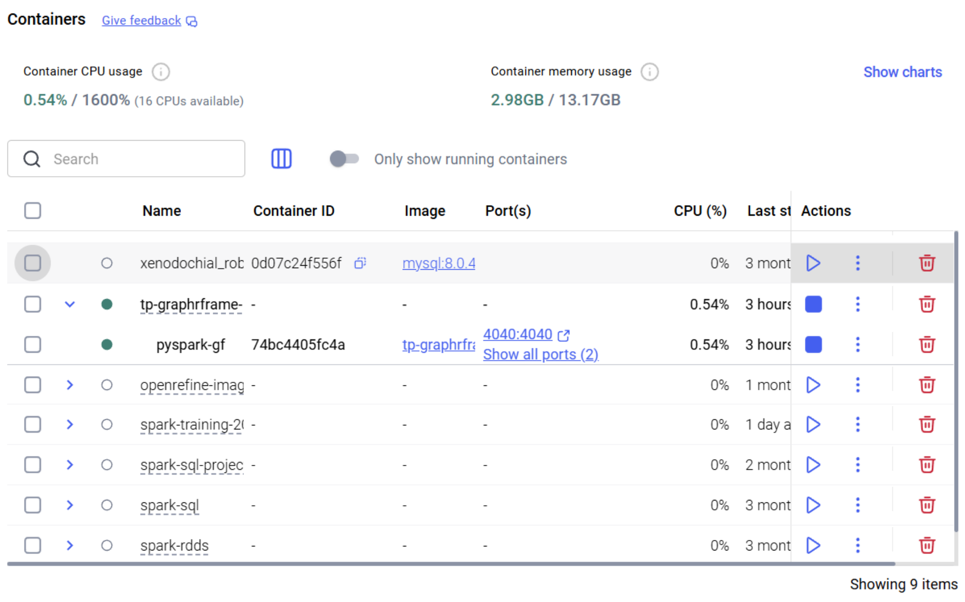
A.1 Environment Setup

The project is designed to run within the provided Docker container.

1. **Start Docker:** Navigate to the folder containing the `docker-compose.yml` file and the notebook. Open a terminal and run:

   ```
   docker-compose up
   ```

2. **Access Jupyter:** Once the container is running, open your browser and navigate to the localhost URL provided in the terminal (usually `http://127.0.0.1:8888`).
3. **Select Kernel:** Open the notebook `TP3_Lovato_Markhovski_Piccoli.ipynb` and ensure the kernel is set to **PySpark**.



Containers Give feedback

Container CPU usage
0.54% / 1600% (16 CPUs available)

Container memory usage
2.98GB / 13.17GB

Show charts

Search

Only show running containers

Name	Container ID	Image	Port(s)	CPU (%)	Last st	Actions
xenodochial_rob	0d07c24f556f	mysql:8.0.4		0%	3 mont	
tp-graphrframe-	-	-	-	0.54%	3 hours	
pyspark-gf	74bc4405fc4a	tp-graphrfra	4040:4040 Show all ports (2)	0.54%	3 hours	
openrefine-imag	-	-	-	0%	1 mont	
spark-training-2(	-	-	-	0%	1 day a	
spark-sql-projec	-	-	-	0%	2 mont	
spark-sql	-	-	-	0%	3 mont	
spark-rdds	-	-	-	0%	3 mont	

Showing 9 items

Figure 3: Docker Container Startup

A.2 Library Installation and Kernel Restart

The visualization tasks (Task 8) require the `python-igraph` and `matplotlib` libraries, which may not be pre-installed in the default `pyspark-gf` container.

1. **Run Installation Cell:** The first code cell in the notebook contains the command:

   ```
   !pip install python-igraph matplotlib pandas
   ```

   Execute this cell to download the necessary packages or open the console via File → New Console for Notebook.

2. **Restart Kernel:** After the installation completes, you must restart the Jupyter kernel for the changes to take effect. Go to: Kernel → Restart Kernel.

A.3 Configuration and Execution

1. **Set the Year Variable:** In the configuration cell (the second cell), there is a global variable named `YEAR`.

   ```
   YEAR = 2018
   ```

   You can choose which dataset, i.e., which year you would like to use for the analysis. Ensure the correct dataset subset is loaded. Changing this variable allows the analysis to be run on different years if data is available.

2. **Run All Cells:** You can now execute the notebook sequentially. Go to Cell → Run All.

A.4 Troubleshooting

Symptom	**Solution**
`ModuleNotFoundError: No module named 'igraph'`	The module was not installed via `pip install` or the kernel was not restarted afterwards.
`AnalysisException: Path does not exist`	Check that the `YEAR` variable is correct and that the CSV file is mounted in the correct path.
Spark Job hangs or crashes during PageRank	The Docker container might be running out of memory. Try reducing the `num_iter` variable, or the dataset, or increasing Docker memory limits.

Table 9: Common Issues and Solutions