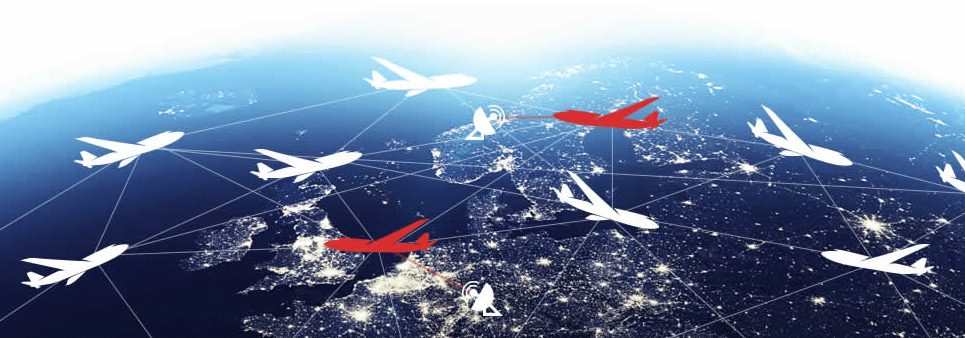
Apache Spark API

Flight Airports

(Average Departure Delays Minimum Spanning Tree)



### *HIGH PERFORMANCE COMPUTING*

**Integrated Master (BSc. + MSc.) of Computer Science and Engineering**

**Faculty of Sciences and Technology of New University of Lisbon**

**(FCT NOVA | FCT/UNL)**

**2018/2019 - 2nd Semester**

### PREPARED BY

**Rúben André Barreiro (Student no. 42648)**

[**r.barreiro@campus.fct.unl.pt**](mailto:r.barreiro@campus.fct.unl.pt)

**Minimum Spanning Tree**

**What is a Minimum Spanning Tree?**

**“A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components.”**

[***Wikipedia*** *-* [*https://en.wikipedia.org/wiki/Minimum\_spanning\_tree*](https://en.wikipedia.org/wiki/Minimum_spanning_tree)]

**Apache Spark**

**“Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since.”**

**“Apache Spark has as its architectural foundation the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API. In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged even though the RDD API is not deprecated. The RDD technology still underlies the Dataset API.”**

[***Wikipedia*** *-* [*https://en.wikipedia.org/wiki/Apache\_Spark*](https://en.wikipedia.org/wiki/Apache_Spark)]

**Project’s Goal**

***Implementing an algorithm to find the best routes in an airports’ network with less average departure delays and compare with another airports’ network with bottleneck departure delays reduced***

**1. Build an undirected graph to represent the average delay of the flights between any two airports (the average must consider all flights between the both airports, disregarding the origin and the destination). The graph’s nodes are thus the airports, and the edges represent direct routes between airports, labelled with the average delays of the routes’ flights.**

**Suggestion: Represent the graph through an adjacency matrix.**

**See** [**https://spark.apache.org/docs/latest/mllib-datatypes.html#distributed-matrix**](https://spark.apache.org/docs/latest/mllib-datatypes.html#distributed-matrix)

**You will have to add the following dependency to file build.gradle:**

* **implementation 'org.apache.spark:spark-mllib\_2.11:2.3.0'**

**2. Compute the graph’s Minimum Spanning Tree (M.S.T.) by implementing the parallel version of Prim’s Algorithm (available from CLIP). The M.S.T. will be the subgraph of the original with the minimum total edge weight (sum of the average delays). Output the M.S.T. and its total edge weight.**

**3. Identify the bottleneck airport, i.e. the airport with higher aggregated delay time (sum of the delays of all routes going out of the airport) from the ones contained in the complement graph of the M.S.T. previously computed.**

**4. Modify the graph to reduce by a given factor the delay time of all routes going out of the selected airport. This factor will be a parameter of your algorithm (received in the command line) and must be a value in ]0, 1[.**

**5. Recompute the M.S.T. and display the changes perceived in the resulting subgraph and on the sum of the total edge weight.**

**Implementation of the Steps**

***Explanation of the implementation of each step of the algorithm, using the Apache Spark API***

**1. The initial bidirectional graph to represent the whole airports’ network was implemented in an adjacency matrix based on Spark’s Coordinate Matrix, with the values containing the average departure delays of the flights. To implement this network was necessary the support of a mapping of the Airports’ IDs to indexes from 1 to the number of airports.**

**2. First, it was generated an initial vertex and verified if it’s a valid initial vertex to start a Minimum Spanning Tree (M.S.T.), after that it was generated a dataset to keep the information about the visited nodes/airports, the current minimum distance (average departure delay) to reach that airport and the antecessor node and generated a new dataset containing all the direct nodes paths, from the initial vertex and the initial it’s marked as visited. Then, iteratively, it was selected the minimum vertex that’s not visited yet and verified if their vertices aren’t reached yet or if are already reached, if have a less cost/distance in comparison with the last verified, until all the airports are marked as visited, using join operations, as also, filter, maps and mapToPair transformations/actions.**

**3. Then, it was computed the complement of the Minimum Spanning Tree (M.S.T.), based on the initial graph and identified the bottleneck airport from it, with the highest aggregate/sum of departure delay, using a reduce transformation.**

**4. The initial graph was modified to be applied the reduce factor to all the routes going out of the bottleneck airport, using map transformations and join operations on the datasets.**

**5. Then, it was calculated a new Minimum Spanning Tree (M.S.T.) from this new graph with the bottleneck airport’s routes cost (average departure delays) reduced, using again join operations, as also, filter, maps and mapToPair transformations/actions.**

Evaluations & Comparisons

**It was performed some experimental tests (most specifically, a total of 3 tests), considering the input files *flights\_4.csv*, *flights\_5.csv* and *flights\_6.csv*. The table of the best obtained results are shown in the following table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Performed Test no.** | **Number of Inputs** | **Number of Airports** | **Runtimes of executions**  **(in hours:minutes:seconds)** |
| **#1** | **14** | **6** | ***00h:01m:14s*** |
| **#2** | **500** | **10** | ***00h:02m:04s*** |
| **#3** | **10,000** | **63** | ***00h:31m:26s*** |

### Table 1: The table of evaluation and comparison of the experimental tests, considering the input files of *flights\_4,.csv flights\_5.csv and flights\_6.csv*

Project’s Repository

**If you are interested in check the *Java and Apache Spark API’s* implementation of the algorithm, as also, the generated outputs from some experimental tests performed, you can check it all in the following *GitHub*’s repository:**

* [**https://github.com/rubenandrebarreiro/apache-spark-api-flight-airports-average-departure-delays-minimum-spanning-tree**](https://github.com/rubenandrebarreiro/apache-spark-api-flight-airports-average-departure-delays-minimum-spanning-tree)

### 

### 

Computer’s Characteristics and Specifications

