POLITECNICO DI MILANO
Master of Science in Computer Engineering

POLO TERRITORIALE DI COMO

Degree Centrality and Superhubs: the lastminute.com Case

Thesis Supervisor: Prof. Sara Comai
Company Tutor: Alessandro Rozza

Candidate:
Andrea Brenna
Student ID: 878929

Academic Year 2019-2020

Rivela il tuo sorriso in una stella se vorrai
Per stasera andrebbe bene anche così

Sommario

L'obiettivo di questa tesi è di analizzare diverse tecniche di centralità applicate sull'enorme rete di trasporto aereo di lastminute.com al fine di estrapolare un elenco di aeroporti che rappresentano gli aeroporti "più trafficati" del mondo soprannominati "superhub". Il primo passo è definire l'ambiente in cui verrà eseguito il progetto e la scelta si baserà sulla piattaforma Google Cloud già utilizzata in azienda. I dati analizzati provengono da diverse interazioni che l'utente può fare sul sito lastminute.com come dati di prenotazione, dati di ricerca e dati dei risultati di ricerca, al fine di avere una mole importante di informazioni e più conoscenza possibile. Il grafo è la migliore struttura dati da sfruttare per questo tipo di problema e le diverse misure di centralità dei grafi verranno sfruttate per ottenere l'elenco dei "superhub". Ci sono molte misure di centralità applicate sui grafi; in questa tesi verranno analizzate cinque diverse tecniche: la classica centralità, eigenvector centrality, Katz centrality, betweeness centrality e una particolare centralità ponderata. La tesi presenta diverse simulazioni al fine di confrontare la misura della centralità menzionata in precedenza in termini di dimensioni del grafico (numero di vertici $|V|$ e bordi $|E|$), complessità computazionale, tempistica di esecuzione, scalabilità e risultati.

Abstract

The goal of this thesis is to analyse different techniques of centrality applied on a huge air transportation network of lastminute.com in order to extrapolate a list of airports that represent the "busiest" airports in the world nicknamed as "superhubs". The first step is to define the environment in which the project will run, and the choice lean on the Google Cloud Platform already used in the company. The analysed data come from different interaction that the user can do on the lastminute.com site such as booking data, search data and search results data, in order to have a lot of information and much more knowledge as possible. The graph is the best data structure to exploit for this kind of problem and the degree centrality measure is used to achieve the extrapolation of the "superhubs" list. There are a lot of degree measures applied on graphs; in this thesis five different techniques will be analysed: the classical degree centrality, the eigenvalue centrality, the Katz centrality, betweeness centrality and a particular weighted centrality. The thesis presents different simulations in order to compare the centrality measure mentioned before in terms of graph size (number of vertices $|V|$ and edges $|E|$), computational complexity, timing of execution, scalability and results.

Ringraziamenti

Desidero innanzitutto ringraziare lastminute.com e nello specifico Alessandro per avermi dato l'opportunità di svolgere questa tesi e di misurarmi con un ambiente lavorativo altamente professionale. Desidero ringraziare la prof. Comai per avermi seguito nella stesura della tesi e per la sua disponibilità nel seguirmi in questo percorso. Ringrazio tutti i miei colleghi per avermi fatto sentire, sin da subito, un membro del team, in particolare Daniele, per avermi sopportato in questi mesi e per avermi accompagnato nel corso di questa esperienza, e Alberto, per il suo supporto ingegneristico e per i suoi preziosi consigli. Un ringraziamento doveroso va sicuramente ai miei compagni di corso con cui ho condiviso anni meravigliosi di studio ma non solo. Vorrei ringraziare tutti i miei amici che mi sono sempre stati vicini dal primo giorno, ma in particolare la mia ragazza Chiara per essere stata la mia ancora di salvezza in tutti i passi di questa avventura. Un ringraziamento speciale alla mia famiglia ed in particolare ai miei genitori per il supporto incondizionato durante questi anni. L'ultimo ringraziamento, ma non per importanza, va a chi non c'è più ma sono certo che il suo sguardo sia sempre stato fisso su di me.
Grazie.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 About lastminute.com

Lastminute.com Group (formerly known as Bravofly Rumbo Group) is a publicly traded multinational Group in the online travel industry and owns a number of brands including:

- lastminute.com: lastminute.com is an online travel and leisure retailer. The company was founded by Lady Lane-Fox and Brent Hoberman to offer late holiday deals online. lastminute.com was acquired by Bravofly Rumbo Group in March 2015.
- Volagratis: Volagratis.com was launched Italy in 2004. Since the pioneering search engine for low cost flights has become a full-service travel provider, offering a range of services including hotels, flights, city breaks, holidays, cruises and car rentals.
- Rumbo: Launched in Spain in 2000, Rumbo also operates websites in other European countries as well as in South America. Rumbo is a full-service travel website, with its offering comprising hotels, flights, city breaks, package holidays and cruises.
- Bravofly: Bravofly is a full-service travel website, integrating a variety of travel products and services, including flights offered by low cost and traditional airlines worldwide. Founded in 2006, Bravofly websites are available in 17 languages and across 40 countries.
- Jetcost: Jetcost is a metasearch website that enables users to search for and compare travel and leisure products from a range of suppliers. Jetcost websites operate in 38 countries across Europe, Asia and America.

- Hotelscan: Hotelscan is a metasearch engine for Hotels, Bed & Breakfasts, Hostels, Apartments and other types of accommodations. Its daily updated database of around 1.3 million properties with data from over 100 online booking websites.
- Weg: weg.de is a German travel website operated by Comvel GmbH, which was founded 2004 in Munich. weg.de, is one of Germany's best-known online travel sites and offers its customers the entire range of travel options, with primary focus on package holidays and all-inclusive vacations.

Every month, the Group reaches across all its websites and mobile apps (in 17 languages and 40 countries) 43 million users that search for and book their travel and leisure experiences. More than 1,200 people work for the Group, which conducts its business through its operational headquarters in Chiasso (Switzerland) and a further 10 offices worldwide.

1.2 Objectives

lastminute.com group is an online travel industry and is interested in understanding the most important airports, called "superhubs", in its air transportation network. Currently there are around 10,000 airports worldwide that make the network of lastminute.com very complex and difficult to explore. The value of importance that lastminute.com gives to "superhubs" is not only related to how well an airport is connected in the network but also how convenient it is, in economic terms, "go through" a specific airport. These specifications are fully combined with the concept of degree centrality of a node in a graph. The degree centrality of a node is one of the main topics regarding the graph theory. Degree is a simple centrality measure that counts how many neighbours a node has and reflects how much a node is important in a network. There are several different techniques in the literature that permit to calculate the degree centrality of a node in a network taking into consideration different aspects related to the structure of the network itself. The goal of this thesis is to give to lastminute.com a list of "superhubs" that combine the concept of traffic and economic convenience by exploring the world of the degree centrality measures and understanding the best technique applicable to the network in question. This thesis is articulated in different steps of analysis, starting from the choice of the environment in which the application should run. The chosen environment is the Google Cloud Platform, offered by Google, a suite of cloud computing services that runs on the same infrastructure that Google uses internally

for its end-user products. Google Cloud Platform offers different kind of products, three of them are fundamental for the development of the project: Cloud Dataproc, Compute Engine and Google Cloud Storage. Their functionalities will be explained in depth in the chapter X. The next step of the analysis is understanding the best data structure to adopt and the most suitable centrality algorithms. The choice of the graph as a data structure is quite natural because permits to represent the air transportation network in a clear and complete way. The airports will be represented as nodes in the graph and a trip between airport A (departure airport) and airport B (arrival airport) will be represent as an edge. In order to enrich the graph, the edges of the network will be "weighted" adding information like price, number of searches per trip, etc. Before starting with the algorithmic part, several jobs were analyzed to get an overview of the most complete on the techniques used and the data structures analyzed. The main goal of this application is to compare different centrality measures in a graph and compare results and performance. The search of centrality measures that were suitable for the network in question and for the extrapolation of the "superhubs" led to the choice of four different techniques: the classical degree centrality, applied to an undirected and unweighted graph; the eigenvector centrality and the Katz centrality that measure of the influence of a node in a network; a weighted centrality based on [1], applied to a undirected and weighted graph, using information about the price and the average number of results giving the searches of a specific period of time. The data manipulated for the creation of the graphs coming from different iteration that a user can do surfing on the lastminute.com site:

- A simple search for a flight with specific regarding: the airports of departure and arrival, that will be represent as a node in the graph; dates, which allows to slice the analysis periods; the number of flight solution proposed by the site, that will be represent as a weight of an edge linking departure and arrival airport.

- A booking of a flight that contains more information than before: the airports of departure and arrival, flight dates, flight duration, price, number of passengers, etc.

The data manipulation phase is a crucial because it is necessary to be able to compress and prepare the data in order to make the creation of the graph and the subsequent analysis as fast as possible. To give an idea of the amount of data we talk about, for a month we have about 500GB of data to manipulate. The last step of analysis is test, on different configuration of

graphs, the results and the performance of the centrality measure listed before. The results highlight how, despite the different techniques adopted and the different approaches in the construction of the graph, the "superhubs" are similar or sometimes equal and concentrated mainly in Europe.

Chapter 2

Environment

This chapter introduces the environment that has been chosen to develop the project, entering in detail the single components, their peculiarities and their usefulness for the final result.

2.1 IDE

2.1.1 Apache Spark

Apache Spark [1] is an open-source distributed general-purpose cluster-computing framework. 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. Apache Spark supports the following four languages that are, Scala, Java, Python and R; in this particular project I use Python. The following are the key features of Apache Spark:

- Polyglot: Spark provides high-level APIs in Java, Scala, Python and R. Spark code can be written in any of these four languages. It provides a shell in Scala and Python. The Scala shell can be accessed through ./bin/spark-shell and Python shell through ./bin/pyspark from the installed directory.

- Speed: Spark runs up to 100 times faster than Hadoop MapReduce for large-scale data processing. Spark is able to achieve this speed through controlled partitioning. It manages data using partitions that help parallelize distributed data processing with minimal network traffic.

- Multiple Formats: Spark supports multiple data sources such as Parquet, JSON, Hive and Cassandra. The Data Sources API provides a

pluggable mechanism for accessing structured data though Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.

- Lazy Evaluation: Apache Spark delays its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed. For transformations, Spark adds them to a DAG of computation and only when the driver requests some data, does this DAG actually gets executed.

- Real Time Computation: Spark's computation is real-time and has less latency because of its in-memory computation. Spark is designed for massive scalability and the Spark team has documented users of the system running production clusters with thousands of nodes and supports several computational models.

- Hadoop Integration: Apache Spark provides smooth compatibility with Hadoop. This is a great boon for all the Big Data engineers who started their careers with Hadoop. Spark is a potential replacement for the MapReduce functions of Hadoop, while Spark has the ability to run on top of an existing Hadoop cluster using YARN for resource scheduling.

- Machine Learning: Spark's MLlib is the machine learning component which is handy when it comes to big data processing. It eradicates the need to use multiple tools, one for processing and one for machine learning. Spark provides data engineers and data scientists with a powerful, unified engine that is both fast and easy to use.

In 2.1 the Apache Spark ecosystem is shown, it presents five different components:

- Spark Core: Spark Core is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an application programming interface (for Java, Python, Scala, and R) centered on the RDD abstraction. This interface mirrors a functional/higher-order model of programming: a "driver" program invokes parallel operations such as map, filter or reduce on an RDD by passing a function to Spark, which then schedules the function's execution in parallel on the cluster. RDDs are immutable and their operations are lazy; fault-tolerance is achieved by keeping track of the "lineage" of each RDD (the sequence of operations that produced it) so that it can be reconstructed in the case of

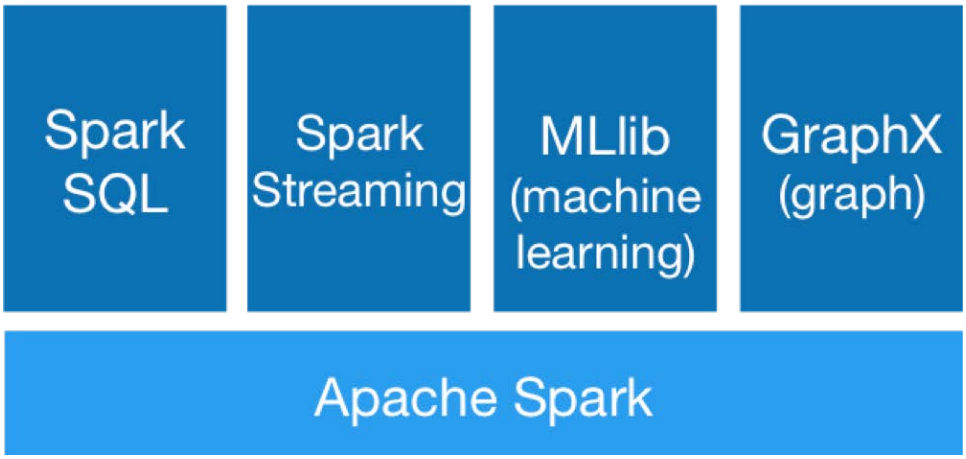

Figure 2.1: Apache Spark ecosystem

data loss. Besides the RDD-oriented functional style of programming, Spark provides two restricted forms of shared variables: broadcast variables reference read-only data that needs to be available on all nodes, while accumulators can be used to program reductions in an imperative style.

- Spark Streaming: Spark Streaming uses Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of lambda architecture. However, this convenience comes with the penalty of latency equal to the mini-batch duration. Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets. In Spark 2.x, a separate technology based on Datasets, called Structured Streaming, that has a higher-level interface is also provided to support streaming. It is used for processing real-time streaming data. Spark SQL: Spark SQL is a component on top of Spark Core that introduced a data abstraction called DataFrames, which provides support for structured and semi-structured data.

- Spark SQL provides a domain-specific language (DSL) to manipulate DataFrames in Scala, Java, or Python. It also provides SQL language support, with command-line interfaces and ODBC/JDBC server. Although DataFrames lack the compile-time type-checking afforded by

RDDs, as of Spark 2.0, the strongly typed DataSet is fully supported by Spark SQL as well. It integrates relational processing with Spark's functional programming API.

- GraphX: GraphX is a distributed graph-processing framework on top of Apache Spark. Because it is based on RDDs, which are immutable, graphs are immutable and thus GraphX is unsuitable for graphs that need to be updated, let alone in a transactional manner like a graph database. GraphX provides two separate APIs for implementation of massively parallel algorithms (such as PageRank): a Pregel abstraction, and a more general MapReduce-style API. Unlike its predecessor Bagel, which was formally deprecated in Spark 1.6, GraphX has full support for property graphs (graphs where properties can be attached to edges and vertices). GraphX can be viewed as being the Spark in-memory version of Apache Giraph, which utilized Hadoop disk-based MapReduce.

- MLlib: Spark MLlib is a distributed machine-learning framework on top of Spark Core that, due in large part to the distributed memory-based Spark architecture, is as much as nine times as fast as the disk-based implementation used by Apache Mahout and scales better than Vowpal Wabbit. Many common machine learning and statistical algorithms have been implemented and are shipped with MLlib which simplifies large scale machine learning pipelines like classification, regression, collaborative filtering, cluster analysis methods, etc.

2.1.2 Python

Python is an interpreted, high-level, general-purpose programming language [2]. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable

names during program execution. Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions; list comprehensions, dictionaries, sets and generator expressions

2.1.3 Graph Library

Python supports various graph libraries.

- iGraph: igraph is a collection of network analysis tools with the emphasis on efficiency, portability and ease of use. igraph is open source and free. igraph can be programmed in R, Python, Mathematica and C/C++ [3].

- Snappy: Snap.py is a Python interface for SNAP [4]. SNAP is a general purpose, high performance system for analysis and manipulation of large networks. SNAP is written in C++ and optimized for maximum performance and compact graph representation. It easily scales to massive networks with hundreds of millions of nodes, and billions of edges. Snap.py provides performance benefits of SNAP, combined with flexibility of Python. Most of the SNAP functionality is available via Snap.py in Python.

- Spark GraphX: GraphX is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

- NetworkX: NetworkX [5] is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. NetworkX provides:

  - tools for the study of the structure and dynamics of social, biological, and infrastructure networks;
  - a standard programming interface and graph implementation that is suitable for many applications;
  - a rapid development environment for collaborative, multidisciplinary projects;

- an interface to existing numerical algorithms and code written in C, C++, and FORTRAN; and
- the ability to painlessly work with large nonstandard data sets.

With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more.

2.2 Vertica

Vertica Systems is an analytic database management software company [6]. The column-oriented Vertica Analytics Platform was designed to manage large, fast-growing volumes of data and provide very fast query performance when used for data warehouses and other query-intensive applications. The product claims to greatly improve query performance over traditional relational database systems, and to provide high availability and exabyte scalability on commodity enterprise servers. Vertica is infrastructure-independent, supporting deployments on multiple cloud platforms (AWS, Google, Azure), on-premises and natively on Hadoop nodes. Its design features include: column-oriented storage organization, massively parallel processing (MPP) architecture, standard SQL interface, in-database machine learning, native integration with open source big data technologies like Apache Kafka and Apache Spark, etc.

2.3 Google Environment - Google Cloud Platform

Google Cloud Platform (GCP) [7], offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search and YouTube. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning. Google Cloud Platform provides Infrastructure as a service (IaaS), Platform as a service (PaaS), and Serverless computing environments. Google Cloud Platform is a part of Google Cloud, which includes the Google Cloud Platform public cloud infrastructure, as well as G Suite, enterprise versions of Android and Chrome OS, and application programming interfaces (APIs) for machine learning and enterprise mapping services. This type of architecture allows to develop the project in a powerful and intelligent way by exploiting the various components that GCP makes available. From the

over 90 products that Google offers, three of these are fundamental to the success of the project: Cloud Dataproc, Compute Engine and Google Cloud Storage (GCS)

2.3.1 Cloud Dataproc

Google Cloud Dataproc (Cloud Dataproc) is a cloud-based managed Spark and Hadoop service offered on Google Cloud Platform. Cloud Dataproc utilizes many Google Cloud Platform technologies such as Google Compute Engine and Google Cloud Storage to offer fully managed clusters running popular data processing frameworks such as Apache Hadoop and Apache Spark. Cloud Dataproc is a Platform as a service (PaaS) product designed to combine the Spark and Hadoop frameworks with many common cloud computing patterns. Cloud Dataproc separates compute and storage, which is a relatively common design for many cloud Hadoop offerings. Cloud Dataproc utilizes Google Compute Engine virtual machines for compute and Google Cloud Storage for file storage. Cloud Dataproc has a set of control and integration mechanisms that coordinate the lifecycle, management, and coordination of clusters. Cloud Dataproc is integrated with the YARN application manager to make managing and using clusters easier. Cloud Dataproc includes many open source packages used for data processing, including items from the Spark and Hadoop ecosystem, and open source tools to connect these frameworks with other Google Cloud Platform products.

2.3.2 Compute Engine

Google Compute Engine (GCE) is the Infrastructure as a Service (IaaS) component of Google Cloud Platform which is built on the global infrastructure that runs Google's search engine, Gmail, YouTube and other services. Google Compute Engine enables users to launch virtual machines (VMs) on demand. VMs can be launched from the standard images or custom images created by users. GCE users must authenticate based on OAuth 2.0 before launching the VMs. Google Compute Engine can be accessed via the Developer Console, RESTful API or command-line interface (CLI). Google provides certain types of machine:

- Standard machine: 3.75 GB of RAM per virtual CPU
- High-memory machine: 6.5 GB of RAM per virtual CPU
- High-CPU machine: 0.9 GB of RAM per virtual CPU
- Shared machine: CPU and RAM are shared between customers

- Memory-optimized machine: greater than 14 GB RAM per vCPU.

Each of these, obviously, has different costs depending on the power of the machine. As part of development, Compute Engine connects various entities called resources, performing different functions. When a virtual machine instance is launched, an instance resource is created that uses other resources, such as disk resources, network resources and image resources.

2.3.3 Google Cloud Storage

Google Cloud Storage is a RESTful online file storage web service for storing and accessing data on Google Cloud Platform infrastructure. The service combines the performance and scalability of Google's cloud with advanced security and sharing capabilities. It is an Infrastructure as a Service (IaaS), comparable to Amazon S3 online storage service. Contrary to Google Drive and according to different service specifications, Google Cloud Storage appears to be more suitable for enterprises. Google Storage offers four storage classes, identical in throughput, latency and durability. The four classes, Multi-Regional Storage, Regional Storage, Nearline Storage, and Coldline Storage differ in their pricing, minimum storage durations, and availability.

2.3.4 Proposal Architecture

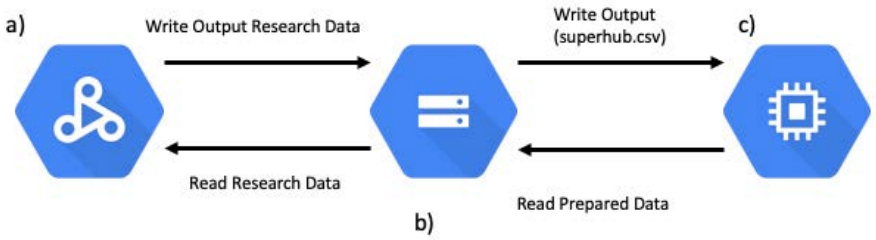

Figure 2.2: Architecture of the project. (a) Cloud Dataproc. (b) Google Cloud Storage. (c) Compute Engine.

In 2.2 the proposal architecture is shown.
Cloud Dataproc is used to take care of reading and preparing data using the Spark libraries; the data is read from the Google Cloud Storage (GCS) on which the result of the data preparation will be written. The Google Cloud Storage (GCS) is used to store all the raw data and all the information useful for the development of the project, containing also additional information useful for the company to the correct operation of all internal operations. Compute Engine is used for many different reasons:

- read the prepared data from the Google Cloud Storage (GCS);
- create graphs using libraries and calculate the degree centrality in order to identify the superhubs;
- write in Google Cloud Storage (GCS) the output (small text file, i.e., .csv or .txt);

The idea behind this architectural proposal is to run the "superhubs" analysis weekly or monthly in order to avoid seasonality and have a list that must be as precise and up-to-date as possible.

Chapter 3

Related Works

This chapter analysis similar works that address the same problem.

3.1 Analysis of the Air Transport Network Characteristics of Major Airports

In [8] the authors analyse the same problem using different software and different methodology. The methodology used in this paper is the Social Network Analysis (SNA) supported by the UCINET, a network analysis software program developed in 2002 by Lin Freeman, Martin Everett and Steve Borgatti.
Social network analysis (SNA) is the process of investigating social structures through the use of networks and graph theory. It characterizes networked structures in terms of nodes (individual actors, people, or things within the network) and the ties, edges, or links (relationships or interactions) that connect them. Examples of social structures commonly visualized through social network analysis include social media networks, memes spread, information circulation, friendship and acquaintance networks, business networks, social networks, collaboration graphs, kinship, disease transmission, and sexual relationships. These networks are often visualized through sociograms in which nodes are represented as points and ties are represented as lines. These visualizations provide a means of qualitatively assessing networks by varying the visual representation of their nodes and edges to reflect attributes of interest. Social network analysis has emerged as a key technique in modern sociology. It has also gained a significant following in anthropology, biology, demography, communication studies, economics, geography, history, information science, organizational studies, political science, social psychology, development studies, sociolinguistics, and computer

science and is now commonly available as a consumer tool. Social network analysis is used extensively in a wide range of applications and disciplines. Some common network analysis applications include data aggregation and mining, network propagation modeling, network modeling and sampling, user attribute and behavior analysis, community-maintained resource support, location-based interaction analysis, social sharing and filtering, recommender systems development, and link prediction and entity resolution. In the private sector, businesses use social network analysis to support activities such as customer interaction and analysis, information system development analysis, marketing, and business intelligence needs (see social media analytics). Some public sector uses include development of leader engagement strategies, analysis of individual and group engagement and media use, and community-based problem solving.
The data analysed in order to calculate the final results coming from three different airline alliances creating a network composed by 1,060 airports and 5,580 routes covering 173 country. The techniques used to extrapolate its "superhubs" are the classical degree centrality and the betweenness centrality, working before on the totality of the world airports and then studying two particular case, the US case and the China case. The final results shown in this paper are different from mine, so considerations must be made. The authors analysed data based on the three biggest airline alliance, that are Star Alliance (composed of 27 members worldwide), Oneworld (composed of 13 members worldwide) and SkyTeam (composed of 19 members worldwide), so with a good coverage of the annual air traffic but considering only the routes these airlines serve during the year. In fact, this study is based on two measure of degree centrality that are not weighted and don't take care of, for example, the number of connections during the period analysed, the cost of the flight or the different trip solutions offered by different airline company. The resulting network is composed by almost all of the world's airports (1,060) but presents a very limited number of connections (5,580) compared with my networks that take care of all the data mentioned before. As already mentioned, the results of this analysis are different from mine because the top30 "superhubs" extrapolated in this work are distributed worldwide (about 80% extra European and only 17% European), whereas my "superhubs" are mostly concentrated in the Europe area. This is due to the fact that lastminute.com customers are mostly European, so the evidence of my analysis is influenced by this.

3.2 Analysing the multilevel structure of the European airport network

In [9] the authors analyse the same problem but take into account only the European Airport Network (EAN), one of the central elements of the World Airport Network (WAN). The data analysed in this paper include all the flights between European cities (including Canary Island and Madeira) in August 2014 covered by the OAG dataset. OAG is an air travel intelligence company that provides accurate, timely and actionable digital information and applications to the world's airlines, airports, government agencies and travel-related service companies. With the world's largest network of air travel data, OAG plays an integral part in helping power a wide variety of industry solutions. They are able to connect their customers to the traveller by providing a full life-cycle of products and they tell the whole story from scheduling and planning, flight status and day-of-travel updates to post journey analysis and on-time performance review. With an unrivalled ability to aggregate complex data sets, OAG delivers real-time flight information insights, compelling visualisations, powerful applications and analytics. OAG is best known for its airline schedules database which holds future and historical flight details for more than 900 airlines and over 4,000 airports. This dataset provides US domestic flight schedules, including airline information (carrier code, name, flight number, etc.) and routing information (airport code, departure and arrival time, stops, number of seats, etc.) from 2008-2019 by year and month. The data structure analysed is a multi-layered network composed by 601 airports and 6,401 connections between nodes coming from a dataset containing 652,291 flights. The limited number of airports is due to a constraint the authors gave to the network that is to consider only the macro airport including all the micro airports associated to it, for example London is indicated as LON (macro airport) including all the nine micro airports of the city, such as London-City (LCY), Gatwick (LGW), London-Heathrow (LHR), London-Luton (LTN), London-Stansted (STN) and so on. The particular network used in this work is exploited to better understand the internal organization of the EAN, composed by three different layers: a core layer; a bridge layer; a periphery layer. It has been used two different measures of degree: a classical degree centrality; a weighted degree centrality based on the number of flights scheduled in August 2014. The core layer contains the nodes belonging to the k-core of maximum k degree, including 69 airports that represent the airports with the highest degree centrality in the network and have an excellent connectivity between them and also with the rest layers of the network. The bridge

layer includes 444 airports that represent a subset of nodes with a good value of degree centrality and a good connectivity between them and also with the core layer. The periphery layer consists of 88 nodes and has low significance in EAN network, with only 27 connections with the bridge and 63 with the core. The results achieved by this paper are similar to mine and also comparable. If we take into consideration only the core layer of the EAN network, it is possible to observe how the airports extract are similar to mine unless there are differences due to the order. In fact, the EAN network constructed can be compared with the networks that will be built due to the fact that, as already mentioned, the lastminute.com market is mostly influenced by European customers. On the other hand, some clarifications and considerations must be made. The EAN network is based on a very small amount of data because take into account only the period of August 2014 in fact the number of connections is very limited. Moreover, the construction of the network doesn't take into consideration the cost of the flight or the different trip solutions offered by different airline company, data that are fundamental in order to have a more precise measure of a weighted centrality. The last weakness of this work is the utilization of only macro airports: this approach gives a solution that will not be precise because "weights" in the same way the micro airports which can be different in size and traffic. For example, London-Heathrow is one of the busiest airport in Europe with 5 terminals and in 2013 served about 72 millions of passengers. In the same period, London-Stansted served about 18 millions of passengers and has only one terminal.

3.3 Robustness analysis metrics for worldwide airport network: A comprehensive study

[10] represents a complete study of worldwide airport network. The authors construct a huge directed and weighted network based on the data extracted from Sabre Airport Data Intelligence (ADI) of 2013. Airport Data Intelligence (ADI) uses bookings made via Global Distribution System (GDS) as its primary data source, and leverages external data sources and proprietary algorithms to estimate direct airline bookings and charter operations to obtain a complete view of scheduling and traffic data. You will be able to obtain:

- Origin airport, destination airport and connecting airport
- Average fare at segment and O&D level

- Point-of-origin airport
- Point-of-sale to zip code level
- Operating and marketing airline
- Booking and cabin-class data
- Future bookings
- Segment traffic divided into local, behind, beyond and bridge

Two different weights are used for this study: geographical distances and the number of passengers travelling between two airports. The resulting network is composed by 3885 nodes and 228080 links, that represents a complete and satisfactory representation of the annual air traffic worldwide. In order to evaluate the strength of a node, the authors analyse the network using 6 different typology of degree, 3 of them used only in my work: weighted degree, where the degree of a node u is the sum of all weights for the incoming and outgoing links of u; betweenness centrality; closeness centrality; eigenvector centrality; Bonacich power centrality; damage. All of these measures are used to "attack" the network in order to truncate some nodes and some connection and then test the robustness of the resulting network. They perform system analysis of the robustness for the worldwide airport network with three different weights (unweighted, distance weighted, and passenger weighted), measured by three different robustness measure. After an exhaustive analysis of time consuming and computational efficiency of all the previous operations, the results are shown. The top 40 airports present an important presence in Asia (35%) and North America (38%) and only in minor part in terms of Europe (18%), whereas my "superhubs" are mostly concentrated in the Europe area. This is due to the fact that last-minute.com customers are mostly European, so the evidence of my analysis is influenced by this.

3.4 Network Centrality of Metro Systems

The last paper analysed, [11], is related to another topology of network, the metro system. As in the air transportation, the metro transportation is a system adapt to exploit the concept of graph and centrality. The authors analyse a total of 28 metro networks, from the smallest, Athens, to the biggest, London, worldwide. The measure of centrality used to calculate the most important nodes in each network is the betweenness centrality, that is

particularly relevant in the case of public transport, in fact, it differs from most centrality indicators. Indeed, the importance does not rely so much on the location of the node as an end point, but on whether or not it is used to join any two other nodes (taking the shortest paths). This kind of paper is not so relevant in order to evaluate my work but is very important to highlight the powerful and ductility of the graph as a data structure and the measure of degree centrality in order to extract the most relevant nodes of a generic network.

Chapter 4

Data Structure & Algorithm

This chapter analyses in detail the data structure and the centrality measures algorithms used and modified in order to reach the desired results.

4.1 Graph

A graph is an abstract data type that is meant to implement the undirected graph and directed graph concepts from mathematics; specifically, the field of graph theory. A graph data structure consists of a finite (and possibly mutable) set of vertices, $|V|$ (also called nodes or points), together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges (also called links or lines), and for a directed graph are also known as arrows. The vertices may be part of the graph structure or may be external entities represented by integer indices or references. A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.). The choice of the graph as a data structure is a rather obvious choice because it allows to represent the airports as nodes and the connection between them as edges. In order to develop the project, I chose a particular kind of directed graph that is the Multi Graph. This kind of graph permits to have multiple arcs, i.e., arcs with the same source and target nodes, that in this particular case represent different flight solutions. Different information is associated with each arc:

- Cost per person;
- Departure date;
- Arrival date;

- Number of hops;
- Flight type;
- Id booking;
- Number of results.

4.2 Centrality Algorithms

In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph.

4.2.1 Degree Centrality

The degree centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information). In the case of a directed network (where ties have direction), we usually define two separate measures of degree centrality, namely indegree and outdegree. Accordingly, indegree is a count of the number of ties directed to the node and outdegree is the number of ties that the node directs to others. When ties are associated to some positive aspects such as friendship or collaboration, indegree is often interpreted as a form of popularity, and outdegree as gregariousness. The degree centrality of a vertex v, for a given graph $G := (V, E)$ with $|V|$ vertices and $|E|$ edges, is defined as

$$C_D(v) = deg(v)$$

Calculating degree centrality for all the nodes in a graph takes $\Theta(V^2)$ in a dense adjacency matrix representation of the graph, and for edges takes $\Theta(E)$ in a sparse matrix representation. The time complexity is $O(|V|)$.
The definition of centrality on the node level can be extended to the whole graph, in which case we are speaking of graph centralization. Let $v*$ be the node with highest degree centrality in G. Let $X := (Y, Z)$ be the $|Y|$ node connected graph that maximizes the following quantity (with $y*$ being the node with highest degree centrality in X):

$$H = \sum_{j=1}^{|Y|} [C_D(y*) - C_D(y_j)]$$

Correspondingly, the degree centralization of the graph is as follows:

$$C_D(G) = \frac{\sum_{i=1}^{|V|}[C_D(v*) - C_D(v_i)]}{H}$$

The value of H is maximized when the graph X contains one central node to which all other nodes are connected (a star graph), and in this case

$$H = (n-1)((n-1)-1) = n^2 - 3n + 2$$

So, for any graph $G := (V, E)$,

$$C_D(G) = \frac{\sum_{i=1}^{|V|}[C_D(v*) - C_D(v_i)]}{|V|^2 - 3|V| + 2}$$

4.2.2 Betweenness Centrality

In graph theory, betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex. Betweenness centrality was devised as a general measure of centrality: it applies to a wide range of problems in network theory, including problems related to social networks, biology, transport and scientific cooperation. Although earlier authors have intuitively described centrality as based on betweenness, Freeman (1977) gave the first formal definition of betweenness centrality. Betweenness centrality finds wide application in network theory; it represents the degree to which nodes stand between each other. For example, in a telecommunications network, a node with higher betweenness centrality would have more control over the network, because more information will pass through that node. The betweenness centrality of a node v is given by the expression:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v. Note that the betweenness centrality of a node scales with the number of pairs of nodes as implied by the summation indices. Therefore, the calculation may be re scaled by dividing through by the number of pairs of nodes not including v,

so that $g \in [0,1]$. The division is done by $(N-1)(N-2)$ for directed graphs and $(N-1)(N-2)/2$ for undirected graphs, where N is the number of nodes in the giant component. Note that this scales for the highest possible value, where one node is crossed by every single shortest path. This is often not the case, and a normalization can be performed without a loss of precision

$$normal(g(v)) = \frac{g(v) - min(g)}{max(g) - min(g)}$$

which results in:

- $max(normal) = 1$
- $min(normal) = 0$

Note that this will always be a scaling from a smaller range into a larger range, so no precision is lost.
On unweighted graphs, calculating betweenness centrality takes $O(|V||E|)$ time using Brandes' algorithm.

4.2.3 Weighted Centrality

The weighted centrality described below is based on [12] Freeman (1978) asserted that the degree of a focal node is the number of adjacencies in a network, i.e. the number of nodes that the focal node is connected to. Degree is a basic indicator and often used as a first step when studying networks (Freeman, 2004; McPherson et al., 2001; Wasserman and Faust, 1994). In an attempt to combine both degree and strength, we use a tuning parameter α, which determines the relative importance of the number of ties compared to tie weights. More specifically, we propose a degree centrality measure, which is the product of the number of nodes that a focal node is connected to, and the aver-age weight to these nodes adjusted by the tuning parameter. We formally propose the following measure:

$$C_D^{wa} = k_i(\frac{s_i}{k_i})^{\alpha} = k_i^{(1-\alpha)} s_i^{\alpha}$$

where:

- $k_i = C_D(i) = \sum_j^N x_{ij}$, where i is the focal node, j represents all other nodes, N is the total number of nodes, and x is defined as 1 if node i is connected to node j, and 0 otherwise;
- $s_i = C_D^w(i) = \sum_j^N w_{ij}$, where w is the weighted adjacency matrix, in which w_{ij} is greater than 0 if the node i is connected to node j, and the value represents the weight of the tie;

- α is a positive tuning parameter that can set according to the research setting and data.

In order to calculate the weighted centrality measure of nodes, I weighted the network using an approach based on the average number of results given a specific request on the lastminute.com site:

$$w_ij = \frac{AveNumberOfResults_{ij}}{AveNunmberOfResults_{kj}(\forall k \neq i)}$$

4.2.4 Eigenvector Centrality

In graph theory, eigenvector centrality (also called eigencentrality) is a measure of the influence of a node in a network. If a node is pointed to by many nodes (which also have high Eigenvector centrality) then that node will have high eigenvector centrality. Relative scores are assigned 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 equal connections to low-scoring nodes. A high eigenvector score means that a node is connected to many nodes who themselves have high scores. Google's PageRank and the Katz centrality are variants of the eigenvector centrality. For a given graph $G := (V, E)$ with $|V|$ vertices let $A = (a_{v,t})$ be the adjacency matrix, i.e. $a_{v,t} = 1$ if vertex v is linked to vertex t, and $a_{v,t} = 0$ otherwise. The relative centrality, x, score of vertex v can be defined as:

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t$$

Where $M(v)$ is a set of the neighbours of v and λ is a constant. With a small rearrangement this can be rewritten in vector notation as the eigenvector equation:

$$\boldsymbol{A}x = \lambda x$$

In general, there will be many different eigenvalues for which a non-zero eigenvector solution exists. However, the additional requirement that all the entries in the eigenvector be non-negative implies (by the Perron-Frobenius theorem) that only the greatest eigenvalue results in the desired centrality measure. The component of the related eigenvector then gives the relative centrality score of the vertex in the network. The eigenvector is only defined up to a common factor, so only the ratios of the centralities of the vertices are well defined. To define an absolute score, one must normalise the eigen vector e.g. such that the sum over all vertices is 1 or the total number of vertices n. Power iteration is one of many eigenvalue algorithms that may

be used to find this dominant eigenvector. Furthermore, this can be generalized so that the entries in A can be real numbers representing connection strengths, as in a stochastic matrix. The time complexity of this centrality is $O(|V|^3)$.

4.2.5 Katz Centrality

In graph theory, the Katz centrality of a node is a measure of centrality in a network. It was introduced by Leo Katz in 1953 and is used to measure the relative degree of influence of an actor (or node) within a social network. Unlike typical centrality measures which consider only the shortest path (the geodesic) between a pair of actors, Katz centrality measures influence by taking into account the total number of walks between a pair of actors. Katz centrality computes the relative influence of a node within a network by measuring the number of the immediate neighbours (first degree nodes) and also all other nodes in the network that connect to the node under consideration through these immediate neighbours. Connections made with distant neighbours are, however, penalized by an attenuation factor α. Each path or connection between a pair of nodes is assigned a weight determined by α and the distance between nodes as α^d . Let A be the adjacency matrix of a network under consideration. Elements (a_{ij}) of A are variables that take a value 1 if a node i is connected to node j and 0 otherwise. The powers of A indicate the presence (or absence) of links between two nodes through intermediaries. If $C_{Katz(i)}$ denotes Katz centrality of a node i, then mathematically:

$$C_{Katz}(i) = \sum_{k=1}^{+\infty} \sum_{j=1}^{n} \alpha^k (A^k)_{ji}$$

Note that the above definition uses the fact that the element at location (I, j) of A^k reflects the total number of k degree connections between nodes i and j. The value of the attenuation factor α has to be chosen such that it is smaller than the reciprocal of the absolute value of the largest eigenvalue of A. In this case the following expression can be used to calculate Katz centrality:

$$\overrightarrow{C_{Katz}} = ((I - \alpha A^T)^{-1} - I)\overrightarrow{I}$$

Here I is the identity matrix, $\overrightarrow{I}$ is a vector of size n (n is the number of nodes) consisting of ones. A^T denotes the transposed matrix of A and $(I-\alpha A^T)^{-1}$ denotes matrix inversion of the term $(I-\alpha A^T)$. Katz centrality can be used to compute centrality in directed networks such as citation networks and the World Wide Web. Katz centrality is more suitable in the

analysis of directed acyclic graphs where traditionally used measures like eigenvector centrality are rendered useless. Katz centrality can also be used in estimating the relative status or influence of actors in a social network. In neuroscience, it is found that Katz centrality correlates with the relative firing rate of neurons in a neural network. Being this measure a variant of eigenvector centrality, it has the same time complexity equal to $O(|V|^3)$.

Chapter 5

Data Analysis

Before starting with the algorithmic part of the project, this chapter presents all the necessary steps before the creation of the graphs and consequent calculation of the "superhubs", from the data acquisition phase to the manipulation and its final aggregation. All these steps are crucial in order to lighten and make the data easier to handle for Dataproc machines and the Spark environment. lastminute.com stores huge amounts of data every day in different locations. The data relating to the number of results produced by a search are stored in Google Cloud Storage (GCS). This type of data will be fundamental in order to create different graphs and be able to take advantage of the weighted centrality explained above. Obviously the data present in these databases are full of additional information not useful to our analysis. For example, the data has about twenty fields but for the purpose of analysis only 4 are fundamental. The data relating to the number of searches is stored in 2 different buckets (one relating to the request and one relating to the response) on Google Cloud Storage according to a well-defined sub-folder structure: a folder for each day of the year within which we find .avro files divided by time ranges. Avro is a data serialization system. Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols.

- Code generation as an optional optimization, only worth implementing for statically typed languages.

Avro relies on schemas. When Avro data is read, the schema used when writing it is always present. This permits each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing. When Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program. If the program reading the data expects a different schema this can be easily resolved, since both schemas are present. Avro schemas are defined with JSON . This facilitates implementation in languages that already have JSON libraries. The size of this raw data before handling is too large and would require hours of processing even on a Google Dataproc cluster. For example, research data occupies around 20 GB of data per day, about 600 GB per month and almost 8 TB for a whole year. The handling and aggregation phases are therefore fundamental. n the following sections the phases described above will be presented in detail, but first we need to introduce two fundamental concepts related to Spark and SparkSQL, Dataset and Dataframe. A Dataset is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). The Dataset API is available in Scala and Java. Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available (i.e. you can access the field of a row by name naturally row.columnName).A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. The DataFrame API is available in Scala, Java, Python, and R. In Scala and Java, a DataFrame is represented by a Dataset of Rows. In the Scala API, DataFrame is simply a type alias of Dataset[Row]. While, in Java API, users need to use Dataset¡Row¿ to represent a DataFrame. This particular data structure combined with the power of SparkSQL will play a fundamental role in all phases of data analysis. The steps of data analysis are represent in figure 5.1.

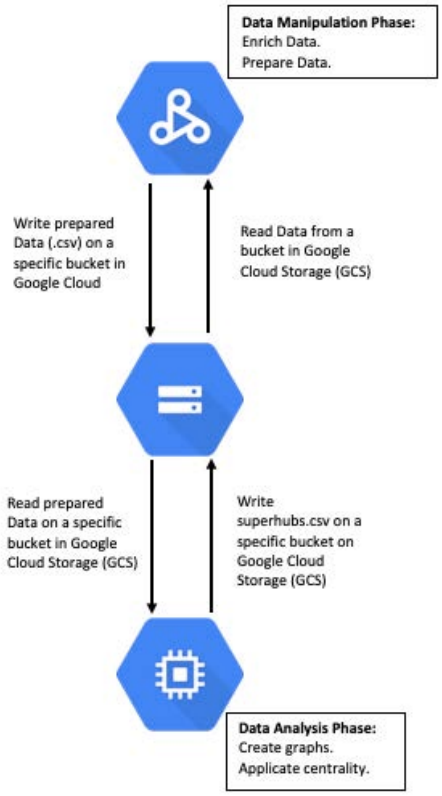

Figure 5.1: Steps of Research Data Analysis

5.1 Data acquisition

The first step is the data acquisition.

The research data, as already said, is stored in two different bucket inside Google Cloud Storage in avro format. The two bucket are named *REQUEST* and *RESPONSE* and between an ID it is possible to associate a request, that a user can do on the lastminute.com site, to the number of total results associate to this request. Also in this case there are a large number of field but only 3 are interesting:

- departure_airport
- arrival_airport
- total_results_number

Downloading this data is simpler than the previous one because using Dataframe and SparkSQL it is possible to acquire the data with a simple python command, e.g.

```
complete_information_dataframe =
sqlContext.read.format('com.databricks.spark.avro')
    .load("GoogleCloudPath")
    .select('field1', 'field2', ..., 'fieldN')
```

The Dataset is represent in Table 5.2. The research data will be saved on

dep_airport	arr_airport	total_results_number
MXP	LHR	324
BCN	DXB	123
...	...	...
...	...	...

Table 5.1: Dataset example of Research Data

the special bucket and then re-acquired to the successive phases.

5.2 Data manipulation

The second step is the data manipulation. This phase is fundamental in order to prepare and enrich the data before the data analysis phase, graphs creation and centrality measure calculations.

In order to prepare the data before create the graphs and calculate the weight centrality based on the average number of results, described in 4.2.3, the acquired Dataframe is enriched with the field ave_results_number_per_search calculated as:

$$ave\_results\_number\_per\_search = \frac{sum\_total\_results\_number}{count\_per\_trip}$$

In the above formula there are two additional fields that enrich the basic dataframe:

- *sum_total_results_number*, that represents the sum of all fields *total_results_number* for a specific trip (departure_airport and arrival_airport) during the period considered.
- *count_per_trip*, that represents the number of occurences for a specific trip (departure_airport and arrival_airport) during the period considered.

The following Python code permits to create the desired data structure:

```
complete_information_dataframe = acquired_dataframe
    .groupBy('departure_airport', 'arrival_airport')
    .agg({'total_results_number': 'sum', '*':'count'})
    .withColumn('ave_results_number_per_search',
        h_udf(sum(total_results_number), count(*))
```

h_udf represents a User-defined function that is a feature of Spark SQL to define new Column-based functions that extend the vocabulary of Spark SQL's DSL for transforming Datasets.
The resulting Dataframe is composed as in Table 5.6.

dep_airport	arr_airport	sum(*)	count(*)	ave_results_number_per_search
MXP	LHR	12000	100	120
BCN	DXB	3000	35	85.71
...	...	...	...	...
MAN	JFK	6540	22	297.27

Table 5.2: Dataframe example with ave_results_number_per_search field

The average number of results per search will be used to determine the weights of the graph.

5.3 Data analysis

The last phase is the data analysis in which the graphs will be created and the centrality measure applied. The research data are used to calculate the weighted centrality explained in 4.2.3 but also to calculate all the other measure explained in Chapter 4. The Python library used is NetworkX. The Dataframe containing the average number of results per trip will be crossed and through special NetworkX functions the nodes will be created and at the same time the edge will be created and weighted according to the weighted centrality formula analyzed in 4.2.3. The NetworkX function used during this phase are:

- creation of a node:

```
graph
    .add_node(id_airport)
```

- creation of a edge:

```
graph
    .add_edge(id_departure_airport,
              id_arrival_airport)
```

- extraction in a dictionary of the classical degree centrality:

```
networkX
    .degree_centrality(graph)
```

- extraction in a dictionary of the betweenness degree centrality:

```
networkX
    . betweenness_centrality(graph)
```

- extraction in a dictionary of the eigenvector centrality:

```
networkX
    . eigenvector_centrality(graph)
```

- extraction in a dictionary of the katz centrality:

```
networkX
    . katz_centrality(graph)
```

The results will be analysed in the next session.

Chapter 6

Scalability

In this chapter will be analysed the scalability of the data acquisition and data manipulation phases.
Scalability is an attribute that describes the ability of a process, network, software or organization to grow and manage increased demand. A system, business or software that is described as scalable has an advantage because it is more adaptable to the changing needs or demands of its users or clients. Scalability is often a sign of stability and competitiveness, as it means the network, system, software or organization is ready to handle the influx of demand, increased productivity, trends, changing needs and even presence or introduction of new competitors. In order to analyze the scalability of this project will be analyzed different amounts of data and different configurations of Google Cloud Platform, in terms of Dataproc (the GCP element that deals with the acquisition and manipulation of the data). Before starting with the analysis of the scalability is important to introduce the concept of Infrastructure as code (IaC) and Terraform [13] and its capabilities. Infrastructure as code is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. Terraform is an open-source infrastructure as code (IaC) software tool created by HashiCorp. Terraform supports a number of cloud infrastructure providers such as Amazon Web Services, IBM Cloud (formerly Bluemix), Google Cloud Platform, Linode, Microsoft Azure, Oracle Cloud Infrastructure, or VMware vSphere as well as OpenStack. In this particular case Terraform is used to build on demand a Dataproc structure on Google Cloud Platform. There are 3 different types of machines that can be instantiated, each with different characteristics in terms of CPU, memory, disk and cost:

- General-purpose machine types: offer the best price-performance ratio

for a variety of workloads. They have up to 96 vCPU and 624GB of memory. They are called n1-standard-X in which X represent the number of vCPUs.

- High-memory machines types: this type is ideal for tasks that require a moderate increase of memory. They have up to 96 vCPU and 624GB of memory. They are called n1-highmem-X in which X represent the number of vCPUs.

- High-CPU machines types: this type is ideal for tasks that require a moderate increase of vCPUs relative to memory. They have up to 96 vCPU and 624GB of memory. They are called n1-highcpu-X in which X represent the number of vCPUs.

The typology chosen for the application is the High-CPU machines types being an CPU intensive application. To test scalability, different High-CPU machines will be used in a master/workers configuration with increasing amounts of research data.

6.1 1 month of data

The first test is based on the research data of May, about 700GB of data. 3 different configuration of master/workers are used:

1. 1 Master and 4 Workers configured as n1-highcpu-16: 109 minutes of execution time.

2. 1 Master configured as n1-highcpu-64 and 4 Workers configured as n1-highcpu-16: 38 minutes of execution time.

3. 1 Master configured as n1-highcpu-64 and 8 Workers configured as n1-highcpu-16: 30 minutes of execution time.

6.2 2 months of data

The second test is based on the research data of May and June, about 1.2 TB of data. The first configuration tested before (6.1) is excluded by this simulation due to the execution times, so 2 different configurations of master/workers are used:

2 1 Master configured as n1-highcpu-64 and 4 Workers configured as n1-highcpu-16: 61 minutes of execution time.

3 1 Master configured as n1-highcpu-64 and 8 Workers configured as n1-highcpu-16: 52 minutes of execution time.

6.3 3 months of data

The last test is based on the research data of May, June and July, about 1.6 TB of data. 2 different configurations of master/workers are used:

2 1 Master configured as n1-highcpu-64 and 4 Workers configured as n1-highcpu-16: 93 minutes of execution time.

3 1 Master configured as n1-highcpu-64 and 8 Workers configured as n1-highcpu-16: 81 minutes of execution time.

The results are summarized in the table 6.1

# months	Size of Data	Conf 1	Conf 2	Conf 3
1	700 GB	109 mins	38 mins	30 mins
2	1.2 TB		61 mins	52 mins
3	1.6 TB		93 mins	81 mins

Table 6.1: Recap Scalability Analysis.

Chapter 7

Results

In this section the degree measures will be analyzed in order to understand the distribution of "superhubs" in the lastminute.com network. The data analyzed and used for the creation of the graphs refer to the May, June and July 2019 quarter.

7.1 Degree Centrality

The first measure analyzed is the classical degree centrality. The results of this analysis are shown in table 7.1. It can be seen that the results of the analysis related to the researches (7.1) present macro airports rather than micro airports; this characteristic is attributable to the fact that during a search on the lastminute.com site the user is usually searching for macro airports and only later he/she will select the micro-airport of interest to him. This peculiarity will be present in all analyzes related to research data. The average of the degree centrality at 2456 airports was 0.0275. London and Paris airport had 18.87 times more connectivity than the average. Adolfo Suarez Barajas Airport, which is the largest airport in Madrid, and Frankfurt International Airport were found to have 15 times higher connectivity than the average. The "superhubs" are concentrated in the European area, while to find an extra European airport you have to go down to the thirtieth position (JFK - New York - John F. Kennedy).

7.2 Betweenness Centrality

The second measure analyzed is the betwenness centrality. The results of this analysis are shown in table 7.2. London and Paris airport had 35 times more connectivity than the average. Adolfo Suarez Barajas Airport, which is

POS	IATA_Code	City	airport_name
1	LON	London	London
2	PAR	Paris	Paris
3	MAD	Madrid	Adolfo Suarez Barajas
4	FRA	Frankfurt	International
5	MIL	Milan	Milan
6	LHR	London	Heatrow
7	BCN	Barcelona	El Prat
8	MAN	Manchester	Manchester
9	ROM	Rome	Rome
10	MUC	Munich	Franz Josef Strauss

Table 7.1: Degree Centrality, based on Research Data. Top 10 'superhubs'.

the largest airport in Madrid, was found to have 14 times higher connectivity than the average. Among the top 30 airports, 23 airports were from the Europe, 4 from Asia, 1 from America and 2 from Oceania.

POS	IATA_Code	City	airport_name
1	LON	London	London
2	PAR	Paris	Paris
3	MAD	Madrid	Adolfo Suarez Barajas
4	MIL	Milan	Milan
5	FRA	Frankfurt	International
6	BKK	Bangkok	Suvarnabhumi International
7	SYD	Sydney	Kingsford Smith
8	LHR	London	Heatrow
9	BCN	Barcelona	El Prat
10	NYC	New York	New York

Table 7.2: Betweenness Centrality, based on Research Data. Top 10 'superhubs'.

7.3 Eigenvector Centrality

The fourth measure analyzed is the Eigenvector centrality. The results of this analysis are shown in table 7.3. London, Paris and Frankfurt international airports had 9 times more connectivity than the average. Adolfo Suarez Barajas Airport, which is the largest airport in Rome, was found

to have 8 times higher connectivity than the average. Among the top 30 airports, 29 airports were from the Europe and only 1 from America.

POS	IATA_Code	City	airport_name
1	LON	London	London
2	PAR	Paris	Paris
3	FRA	Frankfurt	International
4	MAD	Madrid	Adolfo Suarez Barajas
5	BCN	Barcelona	El Prat
6	LHR	London	Heatrow
7	MIL	Milan	Milan
8	MAN	Manchester	Manchester
9	MUC	Munich	Franz Josef Strauss
10	ROM	Rome	Rome

Table 7.3: Eigenvector Centrality, based on Research Data. Top 10 'superhubs'.

7.4 Katz Centrality

The last measure analyzed is the Katz centrality. The results of this analysis are shown in table 7.4. London and Paris airports had 10 times more connectivity than the average. Adolfo Suarez Barajas Airport, which is the largest airport in Rome, was found to have 7 times higher connectivity than the average. Among the top 30 airports, 27 airports were from the Europe, 2 from America and only 1 from Asia.

POS	IATA_Code	City	airport_name
1	LON	London	London
2	PAR	Paris	Paris
3	MAD	Madrid	Adolfo Suarez Barajas
4	FRA	Frankfurt	International
5	MIL	Milan	Milan
6	BCN	Barcelona	El Prat
7	LHR	London	Heatrow
8	MAN	Manchester	Manchester
9	MUC	Munich	Franz Josef Strauss
10	ROM	Rome	Rome

Table 7.4: Katz Centrality, based on Research Data. Top 10 'superhubs'.

7.5 Weighted Centrality

The weighted centrality studied in [12] will be analyzed considering two different value of α and in order to calculate centrality biased towards weight ($\alpha = 1$) and balanced between degree centrality and weight ($\alpha = 0.5$). The results are shown in table 7.5. Although the two analyzes are based on the different balance between degree centrality and weights depending on the alpha used, the results are similar. Considering $\alpha = 0.5$, among the top 30 airports, 20 airports were from the Europe, 5 from Asia, 3 from America and 2 from Oceania. Considering instead $\alpha = 1$, among the top 30 airports, 18 airports were from the Europe, 6 from Asia, 3 from America and 3 from Oceania. The conclusion that can be drawn from these results is that unlike all other measures this presents the most widely distributed list of "superhubs".

POS	$\alpha = 0.5$	$\alpha = 1$
1	PAR	PAR
2	LON	LON
3	MAD	MIL
4	MIL	MAD
5	CPH (Copenhagen)	HKG
6	HKG	CPH
7	CDG	BKK
8	BKK	CDG
9	BCN	SIN (Singapore)
10	ROM	SYD

Table 7.5: Recap Weighted Centrality based on Research Data.

7.6 Recap

The results are summarized in 7.6 for research data. As can be seen the results are almost identical except for some exceptions or changes of position. In the table it is also possible to note that for almost all the results the superhubs belong to the European area and the first positions are always occupied by London, Paris, Madrid, Milan and Frankfurt, not by chance these airports represent the largest hubs in Europe. In fact, for example, London, Paris, Frankfurt - International and Madrid - Alfonso Suarez Barajas are part of the top 50 largest airports in Europe and from these analysis we can deduce that they are also the busiest. If we consider a list of top 30

"superhubs" the percentage of European airports is close to 80% and the remaining part would be occupied by international airports such as Sydney, New York, Bangkok, Dubai, etc.

POS	Degree	Betweenness	Eigenvector	Katz
1	LON	LON	LON	LON
2	PAR	PAR	PAR	PAR
3	MAD	MAD	FRA	MAD
4	FRA	MIL	MAD	FRA
5	MIL	FRA	BCN	MIL
6	LHR	BKK	LHR	BCN
7	BCN	SYD	MIL	LHR
8	MAN	LHR	MAN	MAN
9	ROM	BCN	MUC	MUC
10	MUC	NYC	ROM	ROM

Table 7.6: Recap Centrality based on Research Data.

Chapter 8

Conclusions

In this thesis I have presented different degree centrality measures and an environment capable of handling a large amount of data. The degree centrality measures presented in Chapter 7 underline how the lastminute.com network is conditioned by the European market and how in particular the airports of London, Paris, Madrid, Frankfurt and Milan represent the busiest and most central airports on the world scene. All the lists shown above are calculated considering the May-June-July quarter so the results could be affected by seasonality. The process of calculating or updating the list of "superhubs" is designed to run once a month by acquiring data from the last 3/6 months in order to avoid the seasonal nature of the data as in the case examined.
Another important conclusion can be drawn from the infrastructure with which the project was developed. Google Cloud Platform is a powerful and configurable infrastructure to your liking and with the help of Apache Spark and Python it turns out to be a perfect and efficient infrastructure for handling large amounts of data as in the case of lastminute.com. In this particular case, the configuration of machines used in Chapter 6 highlights an application that is not perfectly scalable but capable of processing almost 2 TB of data in less than two hours.

Bibliography

[1] *Apache Spark*. URL: https://spark.apache.org/.

[2] *Python*. URL: https://www.python.org/.

[3] *iGraph*. URL: https://igraph.org/.

[4] *Snap.py*. URL: https://snap.stanford.edu/snappy/.

[5] *NetworkX*. URL: https://networkx.github.io/.

[6] *Vertica*. URL: https://www.vertica.com/.

[7] *Google Cloud Platform*. URL: https://cloud.google.com/.

[8] Gi Tae YEO Min Geun SONG. "Analysis of the Air Transport Network Characteristics of Major Airports". In: *The Asian Journal of Shipping and Logistics* (2017). DOI: http://dx.doi.org/10.1016/j.ajsl.2017.09.002.

[9] Jose M. Sallan Oriol Lordan. "Analyzing the multilevel structure of the European airport network". In: *Chinese Journal of Aeronautics* (2017). DOI: http://dx.doi.org/10.1016/j.cja.2017.01.013.

[10] Sebastian Wandelt Sun Xiaoqian Volker Gollnick. "Robustness analysis metrics for worldwide airport network: A comprehensive study". In: *Chinese Journal of Aeronautics* (2017). DOI: http://dx.doi.org/10.1016/j.cja.2017.01.010.

[11] Sybil Derrible. "Network Centrality of Metro Systems". In: (2012). DOI: PLoSONE7(7):e40575.doi:10.1371/journal.pone.0040575.

[12] John Skvoretz Tore Opsahla Filip Agneessensb. "Node centrality in weighted networks: Generalizing degree and shortest paths". In: *Social Networks* (2010). DOI: doi:10.1016/j.socnet.2010.03.006.

[13] *Terraform*. URL: https://www.terraform.io/.