CA337 Data Exploration Using Graph Theory

Project

Liam Carroll | 17330013 | liam.carroll36@mail.dcu.ie | DS3

Lecturer: Mark Roantree

Project due date: 17th December 2019

Flight Route Analysis Using Graph Theory

CA337 Graph Theory Project

1. Introduction

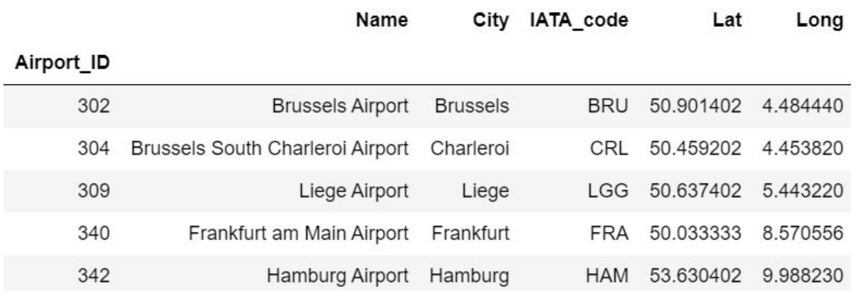
1.1 Background & Motivation

Since the advent of air travel, getting on a plane these days is almost as common as getting on a bus. There are on average 102,465 flights per day worldwide [1]. Some very interesting analysis could be performed on these flight routes, airports and different airline operators around the world using concepts from graph theory. The very nature of flight routes lends itself to be modelled using a graph database over a typical relational model. So in this project, I will endeavour to gather some datasets and design a graph database which models flight routes from different airports around the world. Armed with this database, I will carry out some analysis on this and test some hypotheses which I will formulate later in my report.

1.2 Dataset Description

After deciding that I was going to build a graph database based on flight routes throughout the busiest airports in the world, I had to find a dataset to populate my graph database. I found a website called openflights.org, which contained very large databases of the different airports, airlines and flight routes taken by these airlines all around the world. I downloaded three separate .dat files containing the routes, airports and airlines respectively. The next step was to clean and prepare this data for import into Neo4j. I did this using an open-source tool called OpenRefine and the Python library pandas.

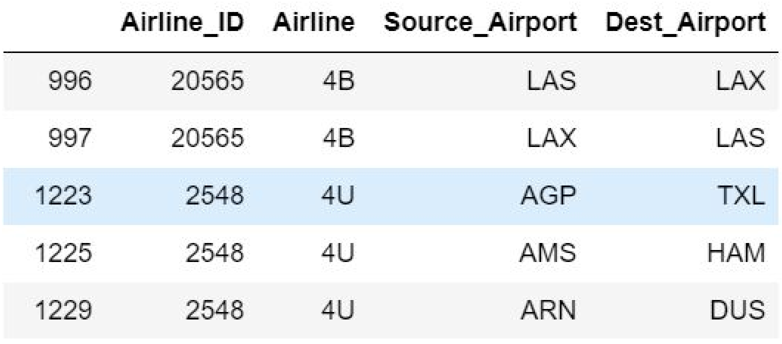
The raw datasets which I downloaded were far too big and would be computationally too expensive to carry out analysis on, so I had to filter them down substantially. Originally, there were over 10,000 airports in the airports dataset but I used OpenRefine to filter out any airports not in the EU or US as that was the region I chose to confine my analysis to. This still returned 2,748 airports so I needed to refine further. I decided to choose only the top 5 airports from each country, provided that they had more than 1,000,000 passengers passing through them per year, and because of the size of the US proportional to the smaller EU countries, I chose the top 10 from there. The layout of the final airport dataframe is shown below:



	Name	City	IATA_code	Lat	Long
Airport_ID					
302	Brussels Airport	Brussels	BRU	50.901402	4.484440
304	Brussels South Charleroi Airport	Charleroi	CRL	50.459202	4.453820
309	Liege Airport	Liege	LGG	50.637402	5.443220
340	Frankfurt am Main Airport	Frankfurt	FRA	50.033333	8.570556
342	Hamburg Airport	Hamburg	HAM	53.630402	9.988230

I got rid of some of the columns in the original dataset as they provided no relevant information for my analysis. The final airport database which would represent the airport nodes in the graph had 98 entries.

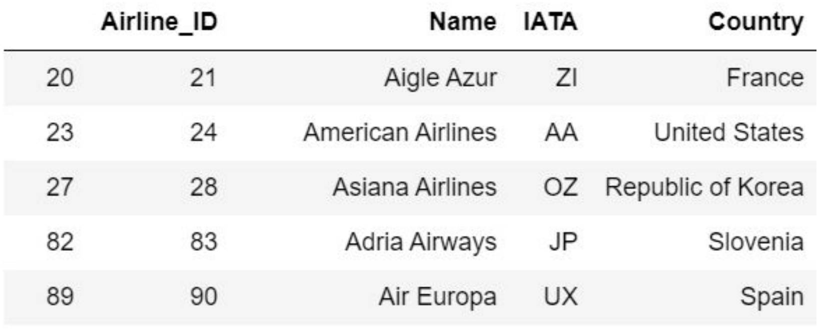
The next step was to filter down the original routes dataset to only contain routes to and from airports in my final airports dataset. The original routes dataset contained 67,663 entries and when I filtered it down to routes whose source and destination were contained in the airports set, I was left with 5,978 entries. The final layout for this dataset is shown below:



	Airline_ID	Airline	Source_Airport	Dest_Airport
996	20565	4B	LAS	LAX
997	20565	4B	LAX	LAS
1223	2548	4U	AGP	TXL
1225	2548	4U	AMS	HAM
1229	2548	4U	ARN	DUS

Again, columns which had no use for my graph database were left out. From the image, you can see that the source and destination airports are represented by their IATA (International Air Transport Association) code.

From the routes dataframe, I constructed a set of all the airline codes which operate the routes in question. This is how I filtered the airlines dataset and it left me with 119 airlines to import. Below, the final airlines dataset is shown, with appropriate columns left out:

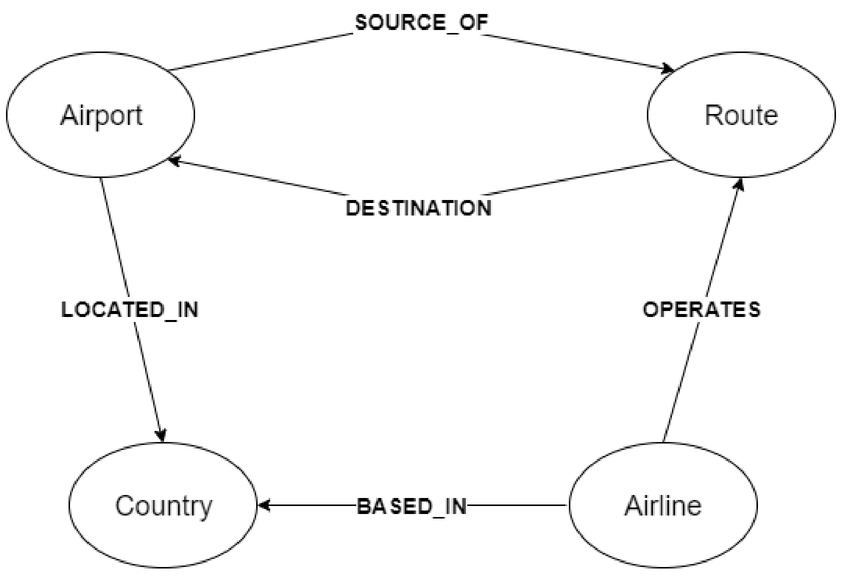


	Airline_ID	Name	IATA	Country
20	21	Aigle Azur	ZI	France
23	24	American Airlines	AA	United States
27	28	Asiana Airlines	OZ	Republic of Korea
82	83	Adria Airways	JP	Slovenia
89	90	Air Europa	UX	Spain

Lastly, to add another dimension to my analysis, I built a countries dataframe from the unique countries in the airport dataset. Before importing these four different types of nodes into the graph, I performed one final check for null values and anomalies using OpenRefine due to the fact that null values cannot be represented by graphs. The full Jupyter Notebook were I carried out this data cleaning can be found here.

1.3 Graph Database Design

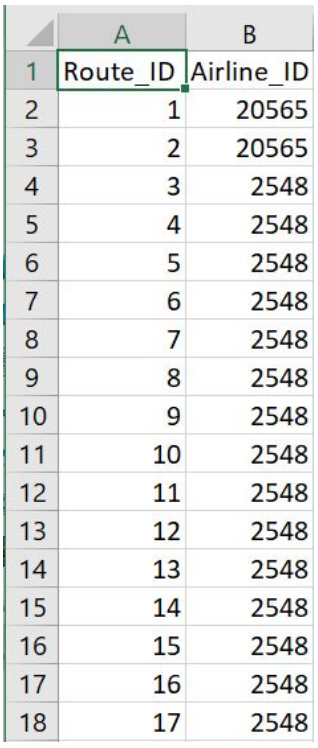
The next step was to consider the relationships which I was going to incorporate in my graph and whether or not they needed properties and if so, what properties they needed. Below is a schema of how I propose to design my graph database:



So, the relationships which I need to build are:

- Airport - [SOURCE_OF] -> Route
- Route - [DESTINATION] -> Airport
- Airport - [LOCATED_IN] -> Country
- Airline - [BASED_IN] -> Country
- Airline - [OPERATES] -> Route

I formed new .csv files which contained the information to link these nodes (each one's unique value) together from the node datasets that I created. The code detailing how I constructed these is contained in the Jupyter Notebook which I linked above. An example of one of the .csv files is shown on the left below.



	A	B
1	Route_ID	Airline_ID
2	1	20565
3	2	20565
4	3	2548
5	4	2548
6	5	2548
7	6	2548
8	7	2548
9	8	2548
10	9	2548
11	10	2548
12	11	2548
13	12	2548
14	13	2548
15	14	2548
16	15	2548
17	16	2548
18	17	2548

The final step after I had prepared all the data for my nodes and relationships was to load this data into Neo4j. So, after placing all the files mentioned above in the 'import' folder, I executed the following code in this file in order to load them into my database.

For each node, I created a constraint on the unique value of that node to make sure the nodes could be easily indexed. Then, I used the LOAD CSV command along with MERGE clause to create the nodes in the graph. In order to reduce the memory overhead of loading the relationships into the graph because there were so many, I implemented the USING PERIODIC COMMIT 500 command to perform a commit after every 500 rows were loaded in.

After I had loaded all the data in, my graph database contained 6,237 nodes and 18,066 relationships. This amount of relationships may pose a computational problem for my computer when running algorithms so some of my analysis may have to be performed on a subgraph but this can still provide meaningful results. The next step is to form some hypotheses about the data.

1.4 Hypotheses

My choice on what airports to include in my graph was based on the fact that I wanted to analyse which airports in Europe served as the main gateways to the United States. A good measure of this would be Betweenness Centrality as it is often used to find nodes which act as a bridge node from one part of the graph to another. So, in my case, find the bridge from the EU to the ten major airports in the US. My initial hypothesis is that London Gatwick will have the highest score for Betweenness Centrality because it serves as a lay-over for many flights transatlantic from Europe. This will be my first research question and I will investigate this using my created graph.

It would also be interesting to see which airport facilitates the most flight routes, or in other words, the busiest or most popular airport in the EU and US. This could be investigated using the Degree Centrality algorithm, which counts the number of in-links and out-links of each node in the graph. Other interesting measures could also be calculated from this, such as the minimum, maximum and average amount of routes which go through all of the airport nodes in the graph. Another reason why this would be a worthwhile piece of analysis is that Neo4j does not currently have a Degree Centrality function so I would have to implement this myself. I will give my two cents on what I think the busiest airport in the EU and US in the second research question section below and then compare this to the results I get from running the algorithm.

2. Research Question 1

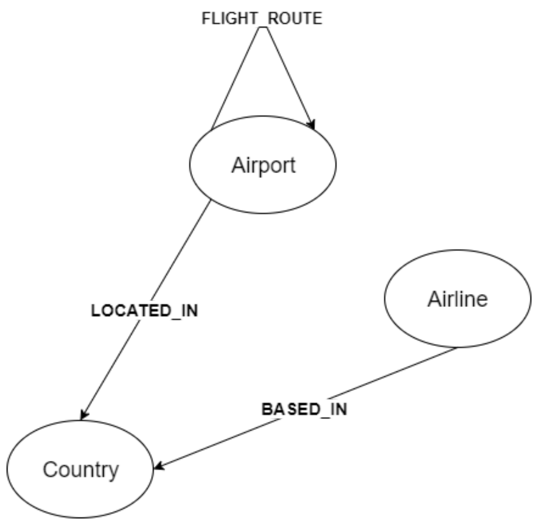
Betweenness Centrality is defined as a way of detecting the amount of influence a node has over the flow of information in a graph [2]. In the case of my graph, we aim to detect which airport has the most influence over the flow of routes through the graph or in other words, which airport is most important at playing the role of a bridge from one part of the graph to another. As I mentioned above, I hypothesise that London Gatwick will attain the highest Betweenness Centrality score for my graph, because a lot of European flights go through there en route to the US.

Due to the fact that the fastest known algorithm for exactly computing betweenness of all the nodes requires at least O(nm) time for unweighted graphs, where n is the number of nodes and m is the number of relationships (for my graph, this would be O(112,677,642)!), I may have to use a subgraph from my original graph to obtain timely results. I'll first try with the full graph however. The code I used to run the algorithm is shown below:

```
CALL algo.betweenness.stream('Airport', 'SOURCE_OF', {direction:'out'})
YIELD nodeId, centrality

MATCH (ap:Airport) WHERE id(ap) = nodeId
RETURN ap.Name AS airport, centrality
ORDER BY centrality DESC;
```

Surprisingly, this returned zero for every airport. Upon investigating, I discovered that this was always going to be the case because I had modelled routes as nodes and not relationships. So, I had to redesign my graph to incorporate routes as relationships between airports to attain meaningful results. The new design of my graph is shown in the schema below. Routes are now treated as relationships between two airports with the relationship properties containing the route information (operator etc.). Airline no longer really serves any purpose in the graph as the information for which airline operates a route is contained in the properties of the FLIGHT_ROUTE relationship but this doesn't matter for our Betweenness Centrality analysis.

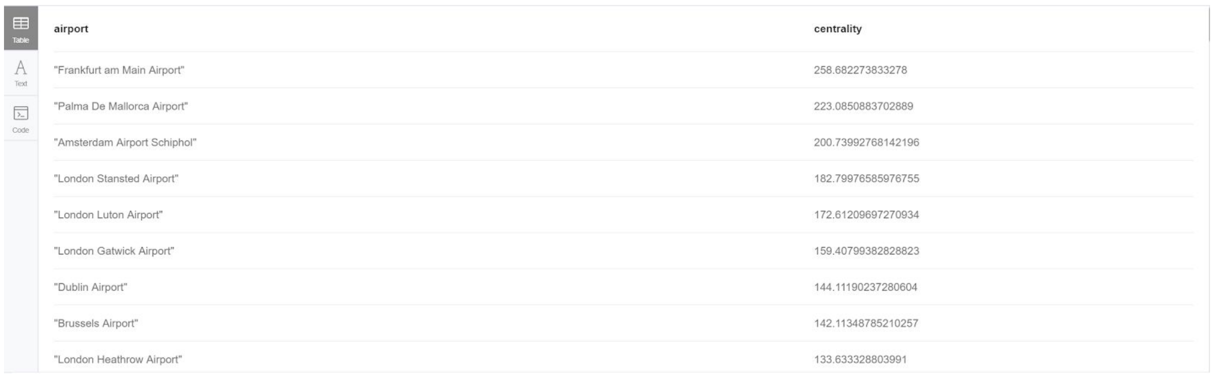


So, re-writing the query to reflect the new graph, we have the following:

```
CALL algo.betweenness.stream('Airport', 'FLIGHT_ROUTE',
{direction:'both'})
YIELD nodeId, centrality

MATCH (ap:Airport) WHERE id(ap) = nodeId
RETURN ap.Name AS airport, centrality
ORDER BY centrality DESC;
```

This query ranks the main brokers of flights in the network and the keyword 'both' for direction implies that the algorithm loads both directions of relationships for analysis, both inwards and outwards. The results are shown below:



airport	centrality
"Frankfurt am Main Airport"	258.682273833278
"Palma De Mallorca Airport"	223.0850883702889
"Amsterdam Airport Schiphol"	200.73992768142196
"London Stansted Airport"	182.79976585976755
"London Luton Airport"	172.61209697270934
"London Gatwick Airport"	159.40799382828823
"Dublin Airport"	144.11190237280604
"Brussels Airport"	142.11348785210257
"London Heathrow Airport"	133.633328803991

Surprisingly, Gatwick did not turn out to have the highest betweenness score. It ranked 6th out of all of the airports, even behind Luton and Stansted, two other London airports. So it's not even the biggest gateway in London, nevermind Europe, according to my analysis! Frankfurt am Main Airport achieved the highest betweenness score and this is understandable as I would imagine a lot of Central European flights pass through it en route elsewhere. Palma De Mallorca ranked second and this was surprising as a lower number of passengers pass through it each year compared to the Madrid and Barcelona airports according to statistics but a possible reason for this could be that Palma de Mallorca operates more international flights than the other two, who could provide more regional and national routes. Amsterdam Schiphol ranked third and again I can understand this as it probably plays a similar role to Frankfurt in terms of connecting Central Europe to the rest of Europe and the US.

Perhaps not so surprisingly due to my graph structure, no US airports featured in the top 10 scores. This can be attributed, however, to the fact that there were a lot more airports from Europe in my graph and therefore, the smaller airports in the EU passing through the larger EU ones would have increased the betweenness score of all of them. All of the US airports were large international airports and so they would not have had to pass through another US airport to reach Europe like the smaller EU ones would have had to reach the US. Now, I will move on to investigate the busiest airports through the Degree Centrality algorithm.

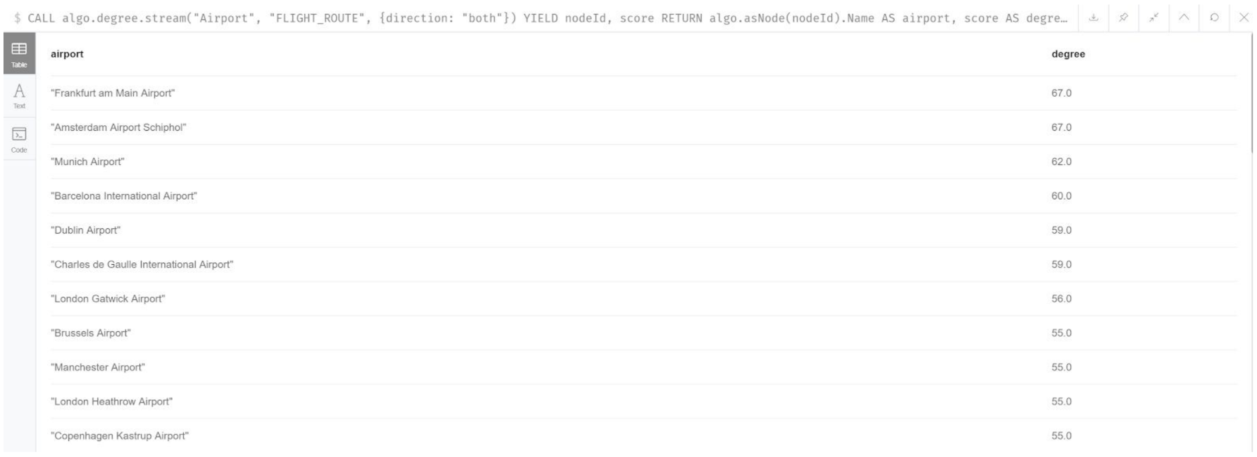
3. Research Question 2

The next piece of analysis I will do on the graph database is to see which airport is the *busiest*, by calculating the number of in-links and out-links for each airport. My initial hypothesis is that Hartsfield-Jackson Atlanta International Airport in the US will score highest because when I was doing research for my project, it was repeatedly quoted as the busiest airport in the world. I also think JFK airport in New York and the three airports in London will score highly because they are in the densely populated cities of New York and London. Due to its high betweenness score, I would not be surprised to see Frankfurt airport score highly here also.

The Degree Centrality will also provide a means of gauging the ‘reach’ of airports. This can be interpreted as: how many other airports can an airport touch right now? To implement this algorithm, I must record the number of incoming and outgoing relationships from all airport nodes in the graph. Conveniently, this can be done using a Neo4j implementation although I did originally think it could be. The code I used to retrieve the degree centrality is shown below:

```
CALL algo.degree.stream("Airport", "FLIGHT_ROUTE", {direction: "both"})
YIELD nodeId, score
RETURN algo.asNode(nodeId).Name AS airport, score AS degree
ORDER BY degree DESC
```

The keyword ‘both’ again signifies to treat the FLIGHT_ROUTE relationship as undirected and therefore, this retrieves both in and out-going flights, and thus the degree centrality of the nodes is returned. The results of this query are shown below:



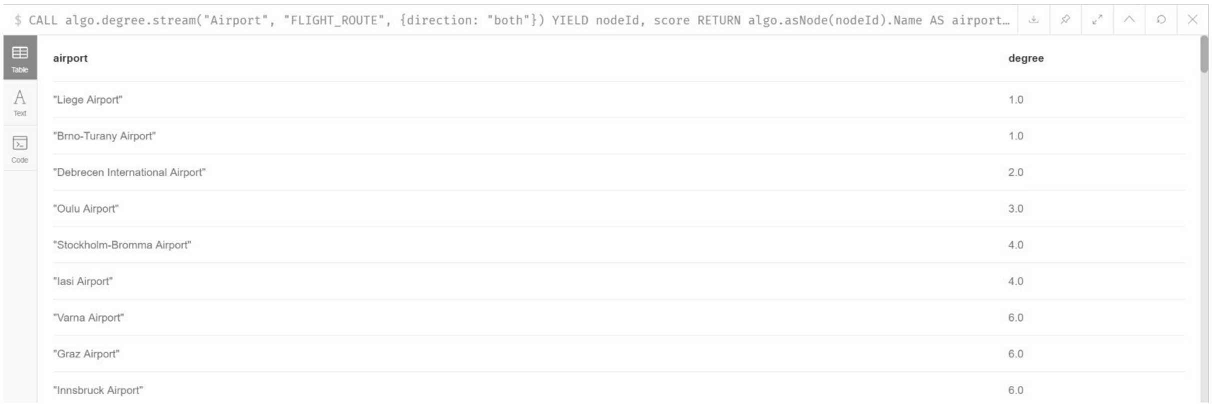
```
$ CALL algo.degree.stream("Airport", "FLIGHT_ROUTE", {direction: "both"}) YIELD nodeId, score RETURN algo.asNode(nodeId).Name AS airport, score AS degre…
```

airport	degree
"Frankfurt am Main Airport"	67.0
"Amsterdam Airport Schiphol"	67.0
"Munich Airport"	62.0
"Barcelona International Airport"	60.0
"Dublin Airport"	59.0
"Charles de Gaulle International Airport"	59.0
"London Gatwick Airport"	56.0
"Brussels Airport"	55.0
"Manchester Airport"	55.0
"London Heathrow Airport"	55.0
"Copenhagen Kastrup Airport"	55.0

It turns out my hypothesis was categorically wrong and upon reflection, I should have foreseen this. There are not enough airports in my graph closely accessible to Atlanta airport compared to how many the EU airports have access to closeby, and consequently it had a low Degree Centrality score of 20.

Again, Frankfurt scored highest in this analysis and so I can rank it as the busiest airport in the network of airports in my graph along with Amsterdam Schipol which also scored 67. Dublin airport was ranked 5th with a Degree Centrality score of 59 so it also is a very busy airport. My hypothesis about Gatwick being the biggest gateway airport in Europe may have been wrong, but it is the busiest airport in the UK according to its Degree Centrality score with Manchester and Heathrow following close behind.

To conclude my analysis of Degree Centrality, I am going to compute the minimum, maximum and mean degree centrality scores for all of the airports in my graph. Obviously, the maximum degree centrality goes to Frankfurt and Amsterdam as shown above but to find the minimum, I simply have to switch the ORDER BY clause of the above code to ASC. When I did this and ran the code, I got the following results:



```
$ CALL algo.degree.stream("Airport", "FLIGHT_ROUTE", {direction: "both"}) YIELD nodeId, score RETURN algo.asNode(nodeId).Name AS airport…
```

airport	degree
"Liege Airport"	1.0
"Brno-Turany Airport"	1.0
"Debrecen International Airport"	2.0
"Oulu Airport"	3.0
"Stockholm-Bromma Airport"	4.0
"Iasi Airport"	4.0
"Varna Airport"	6.0
"Graz Airport"	6.0
"Innsbruck Airport"	6.0

The airports with the lowest Degree Centrality scores were the Liége and Brno-Turany airports located in Belgium and the Czech Republic respectively. They both only had one flight operating which came from or had its destination as one of the other airports in the graph, which is not really a surprising result due to the relative population of these regions compared to the other regions represented by airports in the graph.

The last bit of analysis which I will carry out on the graph is calculating the average Degree Centrality of the airports in the graph. To compute this, I will simply need to divide the number of FLIGHT_ROUTE relationships by the number of airports in the graph. There are 5,977 flight routes and 98 airports so the average Degree Centrality can be computed as:

$$\frac{5{,}977}{98} = 60.9898 \approx 61$$

So, on average, an airport contained in my graph has 61 routes which depart and arrive at that airport. This is quite close to the max Degree Centrality value and therefore tells us that the distribution of degree values for airport nodes is quite negatively skewed and I would anticipate that most airports fall in the 40-67 degree range for number of incoming and outgoing flights. This is likely because I selected the busiest airports from each country and as a result, they would operate a similar enough number of routes.

4. Summary

In conclusion, during this project I learned how to design and build a graph database, converting the data from a typical relational style into a node-and-relationship design with a graph. On top of that, I loaded this data into a graph database, first importing the nodes and then building the relationships. Investigating this graph, I formed two different hypotheses to analyse based on centrality algorithms.

The first was based on the Betweenness Centrality algorithm where I learned that I had to alter the initial design of my graph to obtain meaningful results. From these results, I discovered that the Frankfurt and Amsterdam Schiphol airports were the main ‘bridges’ in the network (i.e. the main gateways for other airports in the EU). I attributed this to their ideal locations in Central Europe and the large populations which they serve.

The second research question which I endeavoured to answer was to find out which of the airports in my graph was the busiest. My initial hypothesis proved to be very wrong as again, Frankfurt and Amsterdam Schiphol were indeed the busiest airports based on how many routes went through each airport. Dublin also scored highly in this algorithm, ranking it as the 5th busiest in my graph. Finally, I computed some summary metrics for degree centrality to give an overall picture of the operating levels of all of the airports in my graph. The scope for further analysis on the graph which I created is huge and many other algorithms could be run on it to answer some other very interesting questions.

References

[1] - *July 25, 2019, Aviation’s busiest day in history* (2019) Available at: https://www.greenclaim.com/news/2019/07/30/july-25-2019-aviations-busiest-day-in-history (Accessed: 10th December 2019)

[2] - *9.2.2. The Betweenness Centrality algorithm* (2016) Available at: https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/betweenness-centrality/index.html (Accessed: 11th December 2019)