Community Structure Analysis of the USA's Airport Network

Federico Fiorini
University of Trento - EIT Digital
Via Sommarive 9, 38123
Povo (TN), Italy
federico.fiorini29@gmail.com

Ana Laura Daniel Diaz
University of Trento - EIT Digital
Via Sommarive 9, 38123
Povo (TN), Italy
anadanield@gmail.com

ABSTRACT
In this paper we present an analysis on the structure of communities of the airports of the United States of America. We use a big dataset containing more than 160M records of commercial flights information in the USA between 1987 and 2015 and use it to identify communities by applying two community detection methods: the walktrap and short-path betweenness algorithms. We detect communities for every year of the available data and use it to analyse how much these communities have changed through the years. Map visualizations and graph plots are used in order to analyse the changes of the communities through the years.

Keywords
Graph Database, Similarity, Community Detection, Network, Random Walks, Shortest-Path Betweenness.

1. INTRODUCTION
Graphs are a very practical way to represent networks. When presented with a dataset about flight information, we used a graph to represent this network of airports and their relationship between each other.

We worked on a dataset which contains records of commercial flights information in the USA between 1987 and 2015. As a way to get interesting insights out of this network of airports and routes, we decided to detect communities of cities by how well connected they are through their departing and arriving flights. Then, we proceeded to detect these communities for every given year of the dataset and in this way be able to analyse how they have evolved throughout the last two decades.

We opted to conduct the detection of communities by applying two opposite approaches to the problem: an agglomerative approach and a divisive one. The two algorithms we used are the **walktrap** algorithm presented by Pons and Latapy in their 'Computing communities in large networks using random walks' paper, and the **shortest-path betweenness** algorithm presented by Girvan and Newman in their 'Finding and evaluating community structure in networks' paper.

So, before carrying out the analysis on the detected communities, we first compared the resulting structure of communities obtained with the two algorithms to find out which of them got better results for this flights dataset. The comparison is made by evaluating the quality of the community partition using the widely accepted **modularity** value.

After choosing the most suitable algorithm, we continue to run this algorithm on every year of the available data. From the detected communities, we build a vector for each of them and compute the distance between them and in this way we get some figures about how different each of the years are.

2. RELATED WORK
Empirical and theoretical studies of networks have been subject of lot of researches in the end of the previous century and early 2000s. Especially a lot of researches have been done in finding community structures that divide a network into groups of nodes with dense connections within the group and sparse connections between them.

In 2003, Girvan and Newman proposed and evaluated three algorithms to find community structures and all three of them have a divisive approach. These algorithms try to calculate a betweenness measure for every edge of the network so the most significative edges can be removed one by one and in this way expose the communities. Two of these algorithms, which are detailed in their 'Finding and evaluating community structure in networks' paper, are the short-path betweenness and the random walk betweenness. In this work, the short-path betweenness algorithm is used to explore the divisive approach of community detection.

Then, in 2005, Pons and Latapy used the basics of the random walks betweenness algorithm to propose their walktrap algorithm which takes an agglomerative approach. They used the concept of the behaviour of a random walk to represent vertices and then compare their similarities to be able to cluster them. The fact that they parted from a divisive algorithm to develop their agglomerative algorithm is rather interesting.

The nature of a network can highly affect the way a network is evaluated and this is why we thought it will be interesting to discover how these two approaches work with a network of cities connected by flight routes.

3. THE DATA
The dataset that will be used to detect communities of cities was obtained from the Bureau of Transportation Statistics of the United States of America and it consists of information on every commercial flight made between 1987 and 2015 within the USA, which adds up to more than 160 million rows of information. The data is heavily focused on the details about the performance of the departure and arrival of the flight, such as estimated times vs actual times, delay causes, cancellations, etc.

However, we've decided to focus only on the information about the origin and destination of the flights and in our previous work 'Storage Systems Comparison between MySQL, MongoDB and

Neo4j using Spark' we detailed how we aggregated this data and stored it in three different databases, one of them being Neo4j, a graph database. The main reason we decided to keep the data in Neo4j it's because it is fairly easy to query to find multiple shortest paths between nodes of a graph and also with a much better performance than the other databases.

3.1 The Graph

The aggregation performed in our previous work resulted in a graph of 330 vertices representing every airport in the USA and over 35 million edges that connected them. The edges, which represent a flight, connect two airports, the origin and destination, and keeps information on how many of these flights were done at a specific day.

But, since we would like to do an analysis of communities through the years, we aggregated this data further by keeping information only about how many flights were done during a whole year. We also decided to ignore the direction of a flight and instead conserve information on the amount of flights made per a route, regardless of the direction. This frequency of flights per route per year help us assign a weight to the graph that will help for the detection of communities. After the yearly aggregation, we ended up with a graph connected by over 115,000 edges.

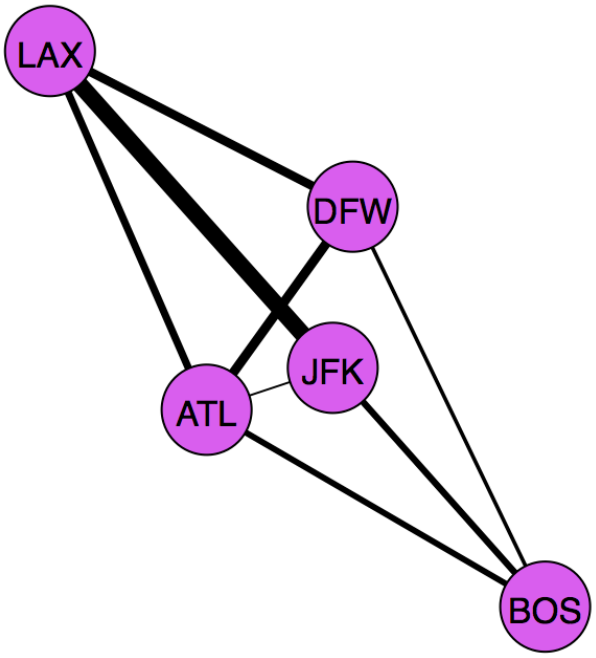

Fig 1. A subset of airports of the flights network. Edge thickness represents the frequency of flights that go through that route.

4. PROBLEM STATEMENT

One of the most relevant features of a network is its community structure, i. e. the organization of its vertices in clusters with high density of edges connecting the vertices within the community and a sparse connection between different clusters. Detecting communities is an important way to extract knowledge and hidden information from a graph, analysing its structure.

Having a network of airports and flights and detecting communities according to the routes frequencies can give us useful information about how well or bad particular cities and areas are connected. Such information can be useful for airlines in order to improve their flight connections.

5. DETECTING COMMUNITIES

To detect communities in the flights graph we evaluated two different algorithms, **walktrap** and **shortest-path betweenness**. The main difference between them is how they start structuring these communities. The walktrap approach is agglomerative, it starts creating many 1-vertex communities and starts merging them based on the vertices' similarity and ends up grouping them all together in one big community containing every vertex. On the other hand, the shortest-path betweenness has a divisive approach, it begins with one big community and starts dividing it by removing edges that have a high *betweenness* and it finalizes with many 1-vertex communities.

We consider it's important to determine which of them results in higher quality communities structure since we plan to run the algorithm many times and identify communities over different states of the graph, depending on which point of time of the available data we would like to analyse.

5.1 Applying the Walktrap Algorithm

Pascal Pons and Matthieu Latapy based their algorithm in the logic that performing random walks in a graph will result in this walk usually getting confined in very connected areas of the graph and they use these random walks to calculate similarities between vertices.

We won't go into deep details but basically they propose to calculate the probability of a vertex i to end up in vertex j while performing a random walk of a given number of steps. They calculate this probability for every other vertex in the graph and use them to represent the vertex i as a vector of probabilities on which said vertex can reach its neighbouring vertices. Once every vertex is identified by a probability vector, the distance between them is calculated to represent their similarity. Then, a hierarchical clustering algorithm is used to merge vector into communities.

The authors suggest that in a network that is globally sparse but dense locally, vertices are grouped into communities that are highly connected between themselves but not so connected with vertices outside their own community, and that these communities could bring a lot of insights about the data. This flights dataset isn't exactly "globally sparse, locally dense", since a lot of airports of the United States are highly connected between each other, however, we think that by accounting the frequency of the route as the weight of the edges, the connection of these communities can be better represented and make this approach still useful to detect said communities.

5.2 Applying the Shortest-Path Betweenness Algorithm

Girvan and Newman, as we saw before, proposed a divisive approach based on the logic that if we find the edge that is more *central* in the graph and we remove it, we divide the graph into different components. If we keep removing the edges which are considered more central in the graph we will end up dividing the graph into disconnected subgraphs: our communities. The authors point out three ways to measure the centrality of an edge and for this paper we have chosen the *shortest-path betweenness* measure: the more shortest-paths pass through an edge, the more that edge is central in the graph.

To calculate the betweenness of an edge we begin by finding all the shortest-paths in the graph. For each vertex in the graph, we find the shortest-path tree that connects that vertex to every other vertex in the graph through the shortest possible path(s). Once the shortest-path tree is defined, we use it to calculate the contribution to betweenness for each edge from this set of paths. Then, after summing the contribution at each step, we have the final shortest-path betweenness values and we can get the edge with the highest value and remove it.

An important step for this algorithm is that at each step after removing an edge we need to re-calculate the shortest-path betweenness in order to find the new edge with the highest value. A different approach could be to calculate the edge betweenness at the beginning, sort the edges by betweenness value and step by step remove them without everytime re-calculating the shortest-path betweenness. This approach is definitely faster but way less accurate. In fact the recalculation step is the most important feature of this algorithm, as far as getting satisfactory results is concerned.

The only difference of running the shortest-path betweenness algorithm on an unweighted graph or on a weighted graph is in the procedure of finding the shortest-path tree. With a weighted graph we need to find the shortest-paths according to the weight of the edges.

5.3 Implementation of the Algorithms

There are several implementations of the shortest-path betweenness algorithm to find communities in a network. We decided to go for a python implementation that uses the *networkx* python library to easily manage graphs and perform operations on them. This library has a useful method *edge_betweenness_centrality* to calculate the betweenness of each edge using Dijkstra algorithm to find the shortest-paths.

As for the walktrap algorithm, we used the original C++ program developed by Pascal Pons. Through a python script, we first load the airports and their routes for any given year from the Neo4j database and build an adjacency matrix, a matrix that keeps track the connection of every airport with each other and how frequent this connection is. This frequency parameter will represent the weight of this edge used when detecting communities with the walktrap algorithm. The existing routes between airports and their weight is then output into a file that will serve as the input edges for the walktrap program which then will output a file with the best partition of communities found and also the modularity value of said partition.

5.4 Adjustments

To adapt the already implemented shortest-path betweenness algorithm to our case we had to change it in order to use properly the weight when calculating the shortest-paths.

In our graph the weight of an edge represents the yearly frequency of the route. However, the edges a shortest-path pass through, by definition, are the ones with a low weight. In fact, usually the weight is interpreted as the cost to pass through the edge and the lower is the weight the better it is. In our case, instead, the more important an edge is, the more high its weight is. In order to make the shortest-paths passing through the high frequency edges we had to "invert" their weight by subtracting the original weight from the maximum weight of all the edges: *edge_weight* = *max_frequency* - *edge_frequency*. We then add 1 in order not to have edges with a weight equal to 0.

One other adjustment that we had to make when applying the algorithms to the network was that we noticed that some years included routes with very low frequency. The flights dataset records every flight ever made by commercial airlines, so we deduced that if a flight had a very low frequency it was probably an exception and this route should not be taken into consideration. A reason that routes with very low frequency could be present in the dataset is because it might have been an emergency landing of a commercial flight that could not reach its original destination due to weather conditions. Because of this, we filtered out routes with less than 52 flights made per year as this will indicate that not even a flight per week is being made through that year. We noticed that the applied algorithm return better modularity value (which will be discussed in the following section) when applying this filter.

5.5 Communities Structure Quality

Like we mentioned, before comparing the structure of the communities of airports through the years we would evaluate the quality of the communities found by each of the applied algorithms: The walktrap, the base short-path betweenness and our adjusted version with the inverted frequency as weight of the edges.

Girvan and Newman defined the **modularity** value to measure the quality of a partition of a network into communities. Said value is obtained by comparing how many edges connect the vertices of a community to other vertices of the same community in relation to the number of the edges that are connected to other communities. So, according to this measure, if the fraction of edges in a network that connect similar vertices that belong to a defined community is better than having random connection between any vertices, it means that the network has been properly divided into quality communities.

In their work, Girvan and Newman also mention that the modularity value of high quality partitions varies from 0.3 to 0.8. Out of the three algorithms we applied to the network of flights, only the walktrap algorithm was able to detect partition of communities with a modularity value greater than 0.3, however, this only was achieved for the partitions of the years from 1987 to 1989. For the rest of the years, the walktrap algorithm found communities with a modularity value that ranged between 0.22 and 0.29.

The short-path betweenness algorithm and its inverted version returned very low modularity values that never were higher than 0.15. Table 1 shows the modularity obtained by each of the algorithms for year 2015 and 2006. If we take into consideration the last 10 years, 2015 has the worst modularity obtained by the walktrap while 2006 has the best one.

Year	Walktrap	Short-Path Betweenness	Short-Path Betweenness Inverted
2006	**0.2963**	0.0630	0.0071
2015	**0.2218**	0.08157	0.0647

It is evident how much better the walktrap algorithm performed compared to the short-path betweenness algorithm and because of this reason we will use the partitions found by the walktrap algorithm when we analyse the structure of the communities through the years.

6. CHANGE THROUGHOUT THE YEARS

6.1 Modularity and Number of Communities

Illustrated by Figure 2, we observed that the quality of the partition that resulted from the walktrap algorithm gradually decreased from 1987 to 2003. The significant increase of 0.7212 that the modularity presented from 2004 to 2006 it presented is rather interesting, however it started to decrease again after that until the present.

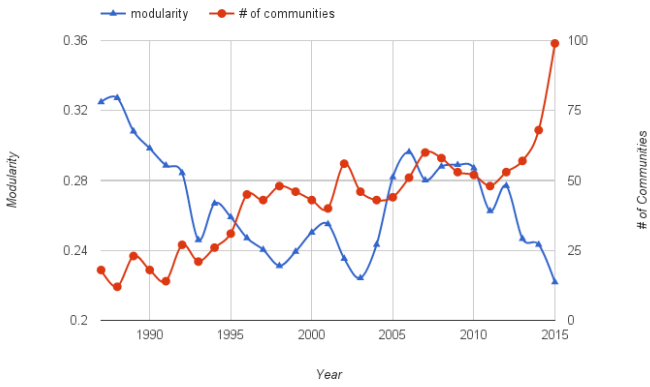

Fig 2. Line chart of the modularity value and number of communities obtained for every year of the dataset (1987 - 2015)

6.2 Visualizing Partitions on a Map

In order to have a better understanding on the communities detected, we placed every airport of the network in a map of the United States of America. Through this map we discovered that the structure of the communities of cities is closely tied to its geographical location. Before applying the algorithm, we speculated that the way the communities will be formed would be based more on how "busy" a city is in terms incoming and outcoming flights and to how many other airports is connected but by looking at the resulting maps we realised that this is not the case.

In Figure 3 we can see how the communities of cities are structured in 2006. As stated before, 2006 was the year that resulted in a partition with the best modularity of the last 10 years. We can visualize five major communities: Two of them are located in single states, in Hawaii (teal colored) and in Alaska (lilac colored), which are two of the states that do not belong to the Contiguous United States of America (CONUS). The other 3 communities are found mainly in the CONUS area: One of them (bright green colored) in the West of the United States; the next one (blue colored), which is the smallest of the communities found in the CONUS, in the Southwest area, with a few of its airports that are also placed in the Midwest; the last one (dark red colored) spans across the Northeast and Southeast.

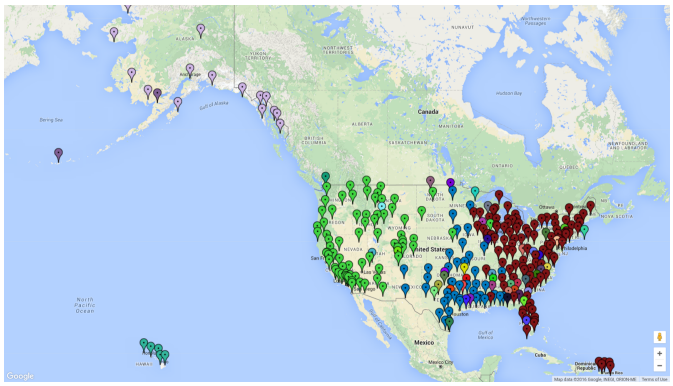

Fig 3. A map of the United States of America with a marker on the active airports of the flown routes of 2006. Markers have different colors depending on the communities they belong to.

One interesting thing to note is how well connected Puerto Rico is with the Northwest and Southeast of the United States even if it's not located in the CONUS area.

6.2.1 Comparing Communities by their Map Visualizations

From Figure 4, one of the things that we noticed was how in the 1980's the airport community of the midwest and southwest was bigger but as time went by, many of these airports starting merging either to the West or the Northeast and Southeast regions of the United States.

Also, by looking at the communities of 2015 we can tell how the ones belonging to the West and to the Southeast and Northeast become less well defined as a lot of different color markers of airports belonging to other minor communities begin to appear within their areas.

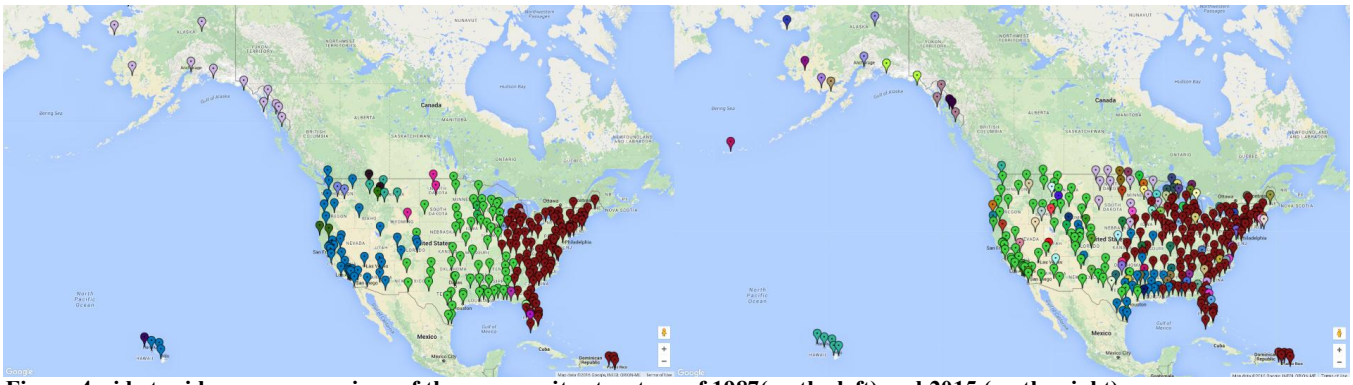

Figure 4. side to side map comparison of the community structure of 1987(on the left) and 2015 (on the right).

By browsing through the resulting visualizations map in a chronological order we were able to observe how the communities of airports and their cities haven't changed that much through the years but like we mentioned, they tend to become less clear when the begin to reach present time. For further visualization we create a gif format images that depicts these yearly changes.

6.3 Visualizing Partitions with Graph Layouts

As explained in section 5.3 about the implementation of the algorithms, we use the *networkx* python library which is used for the "creation, manipulation, and study of the structure, dynamics, and functions of complex networks". One of the advantages of this library is that it let's you plot graphs with the help of the widely used *matplotlib* library.

After creating a graph object in networkx, you can choose the layout in which the nodes should be arranged when displaying the graph. The *shell layout* accepts as a parameter a list of lists of nodes and then place each list of nodes in concentric circles. The parameter that we submitted to this layout was a list of communities and each community was a list of nodes.

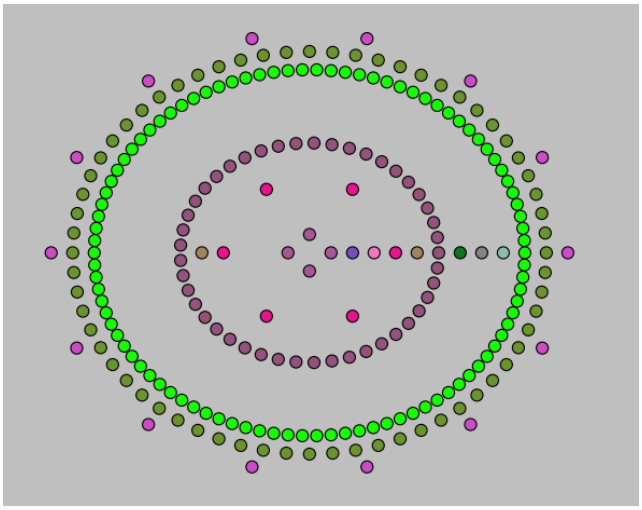

Fig 5. Graph plot of the partitions found in 1987 using networkx's shell layout.

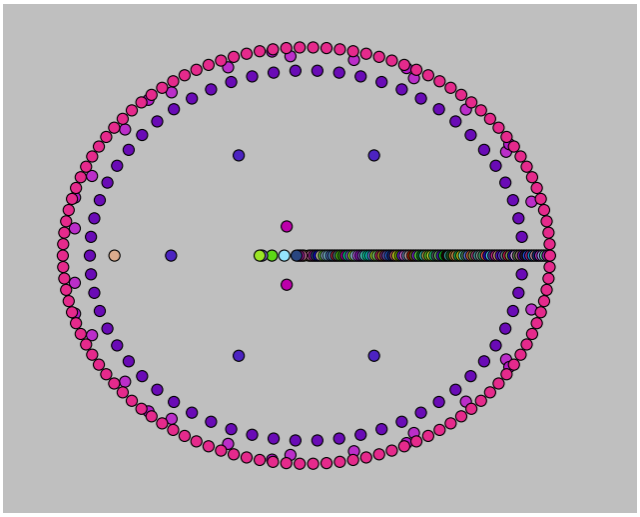

Fig 6. Graph plot of the partitions found in 2015 using networkx's shell layout

By plotting the nodes of the graph in a shell layout we can observe how in 2015 (Fig 6) there are much more single airport communities present than in 1987 (Fig 5). This single airport communities are evident in the plot because since there is only one airport, it is not possible to create a circle path like the shell layout is supposed to create so they all end up forming a very dense horizontal line. This information was not so evident when we analysed the network on a map because of every different color needed for every single vector community.

7. FUTURE WORK

So far we analyzed how the community structure of the network changed through the years using results from different algorithms. An interesting analysis would be to take into consideration smaller fractions of a year, for example three months batches, and detect how the communities change to find seasonal trends.

A different approach could be taken using not only information about the frequency of the routes but rather using the number of passengers of the flights. In this way it would be possible to find migration patterns and flows.

Experimenting with different algorithms for community detection would be interesting as well since during this work we realised that the walktrap and the short-betweenness algorithms did not reach Girvan and Newman's modularity value range of 0.3 to 0.8.

8. CONCLUSION

Over the past 30 years the air travel in the USA has changed a lot not only in the number of airports and flights but also in the community structure of the network. However, dividing this network into good partitions is not trivial task because of the nature of the network itself. Thus, considering only the topology and the traffic is not enough for a spatial network such as the USA Airports Network. Spatial networks, in fact, have a high cost for the long-range connections because physical space is involved. The higher temporal and financial cost has an important role for the passengers and affects the traffic of the network.

After taking a look at the modularity value of the best partitions found by the walktrap algorithm through every year, we can see that it's value has been constantly decreasing. In 1987, the algorithm found a partition that reached a modularity with a value of 0.3272 and from then on it decreased every single following year. By 2015 it returned a partition with a value of 0.2218. The fact that the walktrap algorithm is not able to obtain good quality partitions as the time advances, tells us that the network of airports of the United States of America is becoming very dense both globally and locally.

10. REFERENCES

[1] ACM SIG Proceedings Templates, source: http://www.acm.org/publications/proceedings-template

[2] Detecting Communities in Weighted Graphs, source: https://github.com/kjahan/community

[3] High-productivity software for complex networks, source: https://networkx.github.io/

[4] Storage Systems Comparison between MySQL, MongoDB and Neo4j using Spark, source: https://github.com/federico-fiorini/storage-systems-comparison

[5] Finding and evaluating community structure in networks, source: http://snap.stanford.edu/class/cs224w-readings/newman03community.pd