Group members: Giacomo Gonella, Pietro Picardi, Paolo Bresolin

Meaning and comparison of different centrality measures applied to the USAir97 network and clustering coefficient

Link

The following link is connected to a Google Drive folder containing:

- *inf-USAir97.mtx*: file containing the USAir97 dataset;
- *node_names.txt*: file with the names of the airports. Airport name in line *i* corresponds to node *i* in the graph;
- *Project.ipynb*: Jupyter notebook with the code developed by us for the analysis of the dataset.

Dataset

The dataset is *inf-USAir97*, which contains the flight frequencies of the American company USAir among 332 airports.
The graph is structured in the following way:

- each node represents an airport;
- an edge between two nodes denotes that there are flights provided by the company connecting the two corresponding airports. The graph is undirected, meaning that we do not take into account the direction of the flights, but only the fact that there is a route linking the two airports;
- the graph is weighted: each weight refers to the frequency of flights related to the edge the weight is associated with.

Code and ideas explanation

Evaluations

Preliminary steps

To handle and manipulate the data provided by the dataset inf-USAir97 we used the python library NetworkX.
The dataset is a .mtx file (*sparse matrix format*), to import it in our notebook we used the function *from_scipy_sparse_array* already implemented in NetworkX.

We associated the airport name to each node, thanks to the file created by us *node_names.txt*, first creating a dictionary {*node_index : node_name*} and then using the function *set_node_attributes*.
We implemented numerous ways to compute useful information about graphs, each of them has been made general in order to eventually be used with other graphs.

Degrees and weighted degrees

We created two functions *nodeDegree* and *weightedNodeDegree* to compute the degrees for each node in the graph. It is a really simple information about a node but it can give us interesting discussion points.

Closeness centrality

As we can understand from its name, closeness centrality is a measure of centrality of a node in a network. It is computed as the reciprocal of the sum of the length of the shortest path starting from the node to all the other nodes in the graph, usually, like in our case, it is normalised by multiplying this fraction by the number of nodes minus one:

$$C(x) = \frac{N-1}{\sum_y d(y,x)}$$

We implemented three ways to obtain this data for each node of the graph, two exact algorithms and an approximated one.
The two exact algorithms are pretty simple: they apply the formula written above for every node. The second one considers more nodes for each *for cycle*, faster than the naive approach.
Eppstein-Wang algorithm takes a sample *k* of nodes to approximate the closeness centrality of nodes.
The approximation satisfies our needs, as we can see from the comparison in our code.

Betweenness centrality

The exact betweenness centrality of a node is computed as

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

Where σ_{st} is the total number of shortest path from *s* to *t* and $\sigma_{st}(v)$ is the number of shortest paths that pass through *v*.
Computing this exactly is too computational demanding so we computed it in an approximate way. *approxBetweennessCentr* takes a sample of *k* nodes to compute an expectation of the betweenness centrality of all the nodes.Comparing the approximate results with the exact ones does not give us a super accurate estimate, but for our scope we can use this information without any problems.

Local clustering coefficient

The local clustering coefficient of a node indicates the number of triangles it belongs to.
The approximated version of the algorithm exploits random sampling in order to decrease the rather demanding computational costs, clearly at the expense of accuracy.

Clustering coefficient

The clustering coefficient of a graph corresponds to the number of triangles it contains over the number of all possible triangles.

Random graphs generation

We decided to generate several random graphs, in order to evaluate some of the features that we have computed for the original network *G* also for them. In this way, we can compare the results and try to understand if the results obtained for *G* are significant or not.
There are three different types of randomly generated graphs:

- Erdos-Renyi-Gilbert random graphs with uniformly distributed weights;
- Erdos-Renyi-Gilbert random graphs with weights sampled according to a Normal distribution;
- random graphs with the same unweighted degree sequence of *G*.

The choice of generating different versions of random graphs has been made mainly to understand how they differ in some features values. However, we understood that the class of random graphs with uniformly distributed weights is probably the most suitable of the three to draw a comparison with *G*.
Let us explain more in detail how they are generated.

Uniformly distributed weights

As a first phase, we did not consider weights and generated some random graphs following the Erdos-Renyi-Gilbert model, which works for unweighted graphs. In this way, we got several random networks with the same number of nodes *n* of *G* and a number of edges *m(n, p)*, where *p* is the probability that an edge appears independently of all other events.
Since it is proven that

$$E[m(n,\ p)]\ =\ \binom{n}{2}p$$

(where $E[\cdot]$ represents the expected value of a given expression), in order to obtain

$$E[m(n,\ p)]\ =\ m$$

(where *m* is the number of edges of *G*), we set

$$p\ =\ \frac{m}{\binom{n}{2}}\ .$$

Once obtained an unweighted random graph *G(n, p)* as above, we needed to associate a weight to each edge. As a preliminary step, we computed the minimal weight *min_weight* and the maximal weight *max_weight* in *G*. Then, we gave to each edge of *G(n, p)* a weight sampled uniformly at random in the interval *[min_weight, max_weight)*.
By repeating the above procedure, we were able to generate an arbitrary number of random graphs with similar characteristics that, in our opinion, can be used to make some comparisons with the original network *G*.

Weights sampled according to a Normal distribution

Erdos-Renyi-Gilbert unweighted random graphs were generated as explained in the previous paragraph, but this time we wanted a more sophisticated strategy for assigning a weight to each edge of a random graph *G(n, p)*. Indeed, we computed the mean weight *mean_weight* and the standard deviation *std_weights* of weights in *G* and used them to create samples taken at random following a Normal distribution built using the two parameters just mentioned. Thus, we could assign a sampled weight to each edge of *G(n, p)*. From our point of view, the random networks obtained with this procedure are more similar to the real graph and so a comparison with them can lead to interesting results.

Graphs with same unweighted degree order of *G*

This time we did not use the Erdos-Renyi-Gilbert model, but we tried to maintain the same unweighted degree sequence of *G*. Actually, we also tried to keep the same weighted degree sequence, but we were not able to find an algorithm for doing so.

We started from the real network *G* and then attempted *k* edge swaps. To better understand, consider a general iteration and suppose that the randomly selected edges are *(u, v)* and *(w, z)* with weights, respectively, *w_1* and *w_2*. Say that *E'* is the current set of edges of the random graph *G'* that we are generating. If *(u, z)* and *(v, w)* are not already in *E'*, then *(u, v)* and *(w, z)* are removed from *E'* while *(u, z)* and *(v, w)* are inserted. The weight assigned to *(u, z)* is *w_1* and the one assigned to *(w, z)* is *w_2*.

From what we studied, in case of unweighted graphs, using the algorithm above, we should choose a value of $k \in \Omega(|E|)$, where *E* is the number of edges of *G*. Therefore, we chose a value of *k* as multiple of $|E|$.

The random graphs generated have nodes with the same unweighted degree they have in *G* and the set of weights of the random network is the same as the set of weights of *G*. The differences are in the set of edges and in the correspondence edge-weight.

Random graphs properties computation

After that we had generated *k* random graphs for each of the three types defined above, we computed some of the features previously evaluated for *G* also for those graphs.

The analyses done for each random network are:

- weighted degree of every node;
- exact closeness centrality of every node;
- exact and approximate betweenness centrality of every node;
- exact and approximate local clustering coefficient of each node.

Note that, for running time reasons, we only created 10 graphs per category, but the functions for generating them can take as argument the number of graphs to be generated. Moreover, we provided the function *print_n_in_random_list()*, which can be used to print the features of only *n* graphs chosen uniformly at random among all the generated ones. This can be particularly useful if the user decides to create a large number of random graphs. We commented on the parts of code that use the above function to print the computed values of such selected graphs, but we decided to leave them so that the user can do some tests with the obtained data, if desired.

Monte Carlo approach

General idea

We wanted to understand if some features of *G* were relevant, so we used the Monte Carlo method. Given *n* randomly generated graphs and computed the same measure for all of

them, our goal was to evaluate some *p_values*. Not all *p_values* are discussed in the next paragraphs since once some of them have been explained, all the others are very similar.
The procedure has been applied separately for each different type of randomly generated networks, but then only the more relevant categories are considered.

Example of *p_values*: closeness centralities

We found the node with the largest closeness centrality in *G* and the same for each random graph. Then, we estimated

$$P[largest\ closeness\ in\ random\ graph\ >\ largest\ closeness\ in\ G]$$

as

$$\frac{\#\ random\ graphs\ with\ largest\ closeness\ centrality\ larger\ than\ the\ largest\ closeness\ centrality\ of\ G}{total\ \#\ random\ graphs}$$

Looking at the results, we can observe that:

- random graphs with the same unweighted degree sequence and random graphs with normally distributed weights have very high closeness centralities if compared to the ones of *G*. This is something expected due to how they are defined;
- random graphs with uniformly distributed weights have values similar to *G*. These are the ones that, in our opinion, have to be considered for the discussion. Here the *p_value* is almost 0 and this makes us think that the largest closeness centrality in *G* is a significant result;

Something very similar has been done for computing, as *p_value*:

$$P[mean\ closeness\ in\ random\ graph\ >\ mean\ closeness\ in\ G]$$

Also here, due to how random graphs are generated, we think that the class to be considered for understanding if, on average, the closeness centralities of nodes in *G* are significant, is the one with uniformly distributed weights. As we can see, also in this case the estimated *p_value* is almost 0, so we can say that closeness centralities in *G* are significantly large.
If, instead, we consider:

$$P[lowest\ closeness\ in\ random\ graph\ <\ lowest\ closeness\ in\ G]$$

we will discover that the lowest closeness centrality of *G* is a very significant feature as well.

Other *p_values*: betweenness centralities and degrees

The procedures and results obtained are very similar to the ones of the closeness centrality, but they are better explained in the Jupyter Notebook. Especially, we can observe that:

$$P[largest\ betweenness\ in\ random\ graph\ >\ largest\ betweenness\ in\ G]$$

is usually very low if compared with graphs with uniformly distributed weights.
Also for degrees, some interesting results can be found in the notebook.

Other *p_values*: clustering coefficient

So far we considered only node-level features, but what about graph-level ones?
We computed the clustering coefficient of *G* and all random graphs and put them in comparison. In particular, we wanted to estimate:

$$P[clustering\ coefficient\ random\ graph\ >\ clustering\ coefficient\ G]$$

In this case, we can notice that the above probability is very low for random graphs with uniformly distributed weights and random graphs with normally distributed weights, so the clustering coefficient of *G* can be interpreted as significantly high. This makes sense if we think about the nature of the analysed dataset: if the airline company provides flights from a given airport to other two, it is very likely that the last two are connected by flights as well.

Data Analysis and comparison with the real world

To conclude our report we wanted to create a direct comparison with reality. We printed the 5 top and bottom nodes for each evaluation that we implemented.

We can interpret the weighted degree as the level of traffic of a certain airport. The busiest one is the Chicago O'hare International airport, understandable given its large number of flights (760 daily direct flights to 152 U.S. cities and approximately 95 daily direct flights to 58 international destinations), while the most inactive is the Napaskiak airport, Alaska.

Closeness centrality and betweenness centrality are easy to interpret and they should share some of their top and bottom airports. In fact we find Lambert-St Louis International airport and Stapleton international airport in both lists.

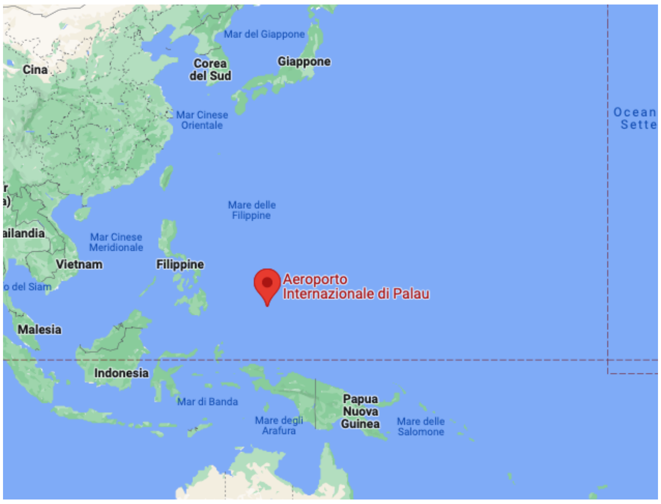
Kansas city international airport is the node with the highest closeness centrality, as we can clearly see it fully reflects this characteristic.

Anchorage international airport is the one with the highest betweenness centrality, this can seem really strange given its location, but Anchorage airport offers an under 10h flight to 90% of the industrial world and it is the second airport in the US for landed weight of cargo aircraft.

Vancouver
WASHINGTON
MONTANA
DAKOTA DEL NORD
MINNESOTA
Ottawa
Montréal
OREGON
IDAHO
WYOMING
DAKOTA DEL SUD
WISCONSIN
MICHIGAN
Toronto
VERMONT
MAINE
NEW YORK
Chicago
NEBRASKA
IOWA
RHODE ISLAND
PENNSYLVANIA
NEVADA
UTAH
Stati Uniti
Aeroporto Internazionale di Kansas City
San Francisco
COLORADO
KANSAS
MISSOURI
DELAWARE
VIRGINIA OCCIDENTALE
KENTUCKY
VIRGINIA
CALIFORNIA
Las Vegas
OKLAHOMA
TENNESSEE
CAROLINA DEL NORD
Los Angeles
ARIZONA
NUOVO MESSICO
ARKANSAS
MISSISSIPPI
CAROLINA DEL SUD
San Diego
ALABAMA
GEORGIA
TEXAS
LOUISIANA
Houston
FLORIDA
Golfo di California
Golfo del Messico
Messico

Aeroporto Internazionale di Anchorage
Canada
Stati Uniti
o Pacifico
ntrionale
Oce
set

Koror airport has the lowest closeness centrality, reasonable being an island in the middle of the Philippines Sea.



The algorithm implemented by us for the betweenness centrality is approximated and the bottom position for this metric are all zeros so taking them into account does not offer us great information.

Through the local clustering coefficient we can get an idea of how much an airport is “replaceable”: those with high clustering coefficient, in case of unavailability, can be easily substituted exploiting their neighbours. On the other hand, a low value means that that airport is both quite isolated and, therefore, hardly replaceable.

Lastly, the clustering coefficient of the graph tells us how robust the network is in case of loss of a certain number of nodes/airports; therefore a high value means good resilience. Comparing the score of the network with the ones of the random graphs we found an expected result: being a resilient network of transportation is an appealing quality that has to be achieved on purpose, therefore our network had to have an high cluster coefficient with respect to its random equivalents.

Machines and tools used to develop the code

We mainly used Google Colab to run our code in a Jupyter Notebook. This is because in that way it has been easier to share modifications, even though no relevant gain in performance was noticeable with respect to our machines.

As package for graph analyses, we used NetworkX.

Contribution of each member of the group

We did not subdivide the work, because we preferred to meet and work together. Indeed, we can say that each member of the group contributed equally to the entire project, because almost all the code has been written together and all the ideas have been discussed together as well. Hence, it is very difficult to assign each idea to a member of the group, but we can assert that each of us contributed with at least one idea per subproblem. However, a rough division based on where each of us developed more code and had more ideas can be the following:

- Giacomo Gonella: clustering coefficient, betweenness centrality, closeness centrality (exact algorithms), random graphs generation (weights in uniform distribution);
- Pietro Picardi: local clustering coefficient, closeness centrality (approximated algorithm), random graphs generation (weights in normal distribution);
- Paolo Bresolin: random graphs generation (same degree sequence), degree and weighted degree, Monte Carlo approach for estimating the p_values.