Experimenting with randomized versions of networks

1. Comparing networks and their randomized versions.

   I recommend to use the same networks as previously, i.e., the small movies/series networks if you need to check code on small networks, and then the airport dataset for interpreting results.

   (a) Using networkx, load the airport dataset.

   (b) Generate an ER random version of it. You can use `gnp_random_graph` or `gnm_random_graph` methods.

   (c) Generate a configuration model version of it, using `expected_degree_graph`

   (d) Compare the network properties of the 3 different versions of the graph, at least the average degree, clustering coefficient, average path length.

   (e) Compute the betweenness, closeness, and PageRank of nodes in the three networks. Compare the largest values between them: are the nodes of highest values the same? Are the highest score similar?

   (f) Plot the distribution of degrees, betweenness and closeness for each network, and compare them. You can use `seaborn` package and functions `distplot` and/or `ecdfplot`, the later being especially useful if you need log scales (`ax.set(xscale="log", yscale="log")`).

   (g) Until now, you have run a single instance of random graph. Each random graph being different, a proper comparison requires to make several runs. For your favorite centrality, compare the change in the values of specific nodes, and the change in the distribution of values between several random graphs generated with the same parameters.

2. Going further : Flavors of the Configuration Model

   We have seen that there are several ways to generate random graphs, usually at least a *soft* and a *sharp* versions. Let's explore the differences in more details.

   (a) Check the documentation of the `configuration_model`, `expected_degree_graph` and `random_degree_sequence_graph`. Do you see the difference? What are the types returned by these functions, `Graph` or `MultiGraph`?

   (b) Try to generate graphs of increasing density. For the sake of simplicity, you can generate uniform distributions, such as the degree raise from 0 to $N - 1$.

   (c) Do you run into problems with some methods?

   (d) Check the degree distribution of the models, are they identical?

   (e) Check the number of self-loops and multiple edges. How does it evolves with the density?

   (f) Generate a power law degree sequence, either manually, or using the `random_powerlaw_tree_sequence` method, or extracting it from a real graph. Generate random graphs using the 3 variants.

   (g) Check using network metrics how different the generated networks are. Most scores are not well defined for MultiGraphs, if you transform it into a simple graph first, how does it affect the results?