Experimenting with Spreading Processes

Several libraries exist to do diffusion on networks. I propose to use `ndlib`, that can be installed using `pip`.
You can have a look at the tutorial to get started: https://ndlib.readthedocs.io/en/latest/tutorial.html.

1. Spreading on a real network
   (a) Initialize a SIR model on the airport network using

   ```
   import ndlib.models.epidemics as ep
   model = ep.SIRModel(g)
   ```

   (b) Create and initialize a custom configuration for your model, following https://ndlib.readthedocs.io/en/latest/tutorial.html#configure-the-simulation. Start with 1% of nodes infected, and a same value for β and γ.

   (c) Still following the tutorial, run the simulation, plot the evolution of the fraction of nodes that are Infected and Removed. Note: you need to run the command `output_notebook()` once before `show()` in order for the plot to appear in a notebook. Adjust the number of steps to see the process until stabilization.

   (d) Run the simulation a few times and observe how much the results differ from one run to the next.

   Tip: if you want, you can "play" the evolution of the diffusion on the graph by doing 1 or a few iterations (the node status are stored in the networkx graph object), and plotting the graph with something like: `nx.draw_networkx(g,pos=coordinates,node_color=list(model.status.values()),with_labels=False)`
   Where coordinates corresopnd to coordinates extracted as in the spatial network exercise.

   Optional: To properly study the results of simulations, you should analyze results averaged over multiple simulations. If you see how to do such tests with python, you can extract the results of a simulation using for instance `trends[0]["trends"]["node_count"][1]`, and create averaged plots, or learn how to use ndlib build-in tools.

2. Spreading compared with synthetic networks
   (a) Create an ER random graph with the same number of nodes and edges than the airport network. Run an **SIS** model on it (check the names of parameters at https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SIS.html), initially with `beta=0.1` and `lambda=0.1`.

   (b) Let's say that we identified a virus with `lambda=0.1`, and that we can use policies to reduce β. What is the value of β above which there will be an outbreak, in theory, on this ER random network ?. Check that if the value is below it, the diffusion stops early, and that, if it is above, the diffusion reach a stable point at a value which depends only on the parameter and the average degree.

   (c) Test the same model on the original network, with values of β below and above the threshold. What do you observe?

3. Optimal node removal
   (a) Let's assume that we can vaccinate some limited fraction of nodes. In practice, vaccinated nodes are removed from the network before running the SIS model. Starting with `beta=0.1` and `lambda=0.1`, what fraction of nodes to you need to vaccinate so that the virus infect less than 30% of non-vaccindated nodes? You can remove random nodes with:

```
g2=g.copy()
g2.remove_nodes_from(random.sample(list(g.nodes()), X))
```

(b) Same question but removing nodes of largest degrees. You can for instance use the following code to sort airports by degree: `top_nodes = sorted(dict(g.degree).items(),key=lambda x:x[1],reverse=True)` (be careful to then extract the nodes names as a list)

(c) Same question but removing nodes of largest betweenness

4. Going further: Dependence on initial conditions

   (a) Using `cfg.add_model_initial_configuration("Infected", infected_nodes)`, you can choose which nodes are infected in the beginning. Start by infecting the 5 nodes of highest degrees, then the 5 nodes of lowest degrees. Observe the differences.

   (b) Start by infecting 5 nodes in the same country, and then 5 random nodes. Observe the differences.