COM-308 – Internet Analytics – EPFL – Spring 2022

Aswin Suresh, Arnout Devos & Matthias Grossglauser

2. Networks: structure, evolution, processes

⚠ This lab does not require the use of Spark; your own machine (laptop/...) is sufficient.

Objectives

Graphs are at the root of many modern applications and you are certainly using many in your everyday life. Common examples of networks include *social networks* used for advertising and recommendation systems, or the *web graph* searched millions of times a day whenever anyone searches on the web. The goals of this lab are the following:

1. You will first extract and gain insights on the properties of large networks.
2. You will also explore sampling methods to estimate some of these properties when you don't have access to the whole graph topology. These methods are often used for crawling social networks.
3. You will then simulate stochastic processes over networks and develop methods to control how they propagate.
4. You will finally explore a powerful method to rank nodes in a network: the PageRank algorithm.

Deliverables

Four template notebooks

```
ix-lab2-notebook-<1–4>.ipynb
```

are provided to you in the `assignment/` folder with a predefined structure for each part of the lab. **The four notebooks should be submitted to Moodle in HTML format** before the deadline (in ~4 weeks, the precise date will be communicated on Moodle). You can easily export your notebooks in HTML using the menu command:

```
File > Download as > HTML (.html)
```

Your final notebooks should contain:

- the **code** to solve the exercises,
- the corresponding **output figures**,
- your **interpretations** and **justifications** written in markdown cells.

Also, note that:

- Only one submission per group is required.
- You are allowed to use any Python library, as long as it is not explicitly mentioned otherwise. Lab 1 can give you an idea of useful libraries.
- Please properly comment your code. Code readability will be considered for grading.
- To avoid long cells of code in the notebook, it is recommended to embed long python functions and classes in a separate Python module. Take lab 1 with its modules in `modules/` as an example. **Don't forget to hand in your module if that is the case.**
- In some exercises, you are required to come up with your own methods to solve various problems. Be creative and clearly motivate and explain your methods. Creativity and clarity will be considered for grading.

2.1 Exploration of real networks

In this section, we will explore the structure of two real (but anonymized) networks:

```
    network1.csv
    network2.csv
```

located in the `data/` folder. Both files are comma-separated edge lists where each line corresponds to an edge of the graph. For example, the line

```
    i,j
```

indicates that there is an edge between node `i` and node `j` in the graph. Additionally, all lines starting with a hash, `#`, should be ignored. They are comments providing information about the dataset (if the graph is directed, ...).

Exercise 2.1
Your first task is to explore `network1` and analyze its properties.

1. Load the graph in your favorite data structure. (E.g., with `networkx`)
2. How many nodes and edges does the graph contain? ■

Exercise 2.2 Node degree distribution
A convenient way to start exploring the structure of a graph is through its node degree distribution. Compute the node degree distribution of `network1` and visualize it.

1. What kind of plot do you think is useful for this task?
2. Can you list some of the properties of the degree distribution?
3. Is it close to a well-known probability distribution? If so, provide an estimation of its parameter(s)? ■

Exercise 2.3 Giant component
Real networks are generally not connected, but one connected component, *i.e.,* the giant component, is usually much larger than the others.

1. Count the number of connected components in `network1`.
2. Is there a giant component ? If so, what is its size? ■

Exercise 2.4 Short paths & Small-worlds
In his book “Six Degrees” (2006), Duncan Watts explains the small-world effect by the now famous claim that every human on our planet is connected by an average of only “*six degrees of separation*”. Do you think that `network1` is a small-world? Justify your answer.

Hint:

- *What is the distribution of path lengths between any two nodes in the network?*
- *How fast is the network growing? I.e. what is the average number of nodes reachable within a distance r (in number of hops) of any other node in the network?* ■

Exercise 2.5 Network comparison
You will now focus on a second real network `network2.csv` located in the `data/` folder.

1. Using the properties computed in Exercises 2.1, 2.2, 2.3 and 2.4 explore the structure of this network.
2. How does this network differ from the previous one? ■

Exercise 2.6 Network identification
Among the two networks you analyzed:

- One of them is the network of roads around the city of New-York, *i.e.,* nodes are intersections and edges are roads between them.
- The other one corresponds to a subgraph of routers comprising the Internet, *i.e.,* nodes are routers and edges are physical links between them.

Can you guess which one is which? Justify your answer. ■

2.2 Network sampling

All the metrics analyzed in the previous section are straightforward to compute when we have full access to the network. However, this is hardly the case in practice. In the following part, we will explore a method to estimate some of these properties on large networks using sampling. This method is often used to analyze social networks.

2.2.1 Age of Facebook users

You are curious about the age distribution of the users on *Facebook*. As you cannot have this information directly from *Facebook* itself, you decide to write a small program that allows you to crawl the network. Fortunately, *Facebook* provides an API[1] to extract information about a node in the network through HTTP requests of the form (accessible on EPFL network/VPN):

```
http://iccluster028.iccluster.epfl.ch:5050/v1.0/facebook?user=<id>
```

where `<id>` is the id of the user you are looking for. If the id is valid, then such a request returns a `json` file formatted as follows:

```
{
    "age": 24,
    "friends": ["id1", "id2", "id3", ...]
}
```

Exercise 2.7 Random walk on the *Facebook* network
Crawl the *Facebook* graph to estimate the average age of users in the social network using the random walk approach of Algorithm 2.1, starting from node

`"a5771bce93e200c36f7cd9dfd0e5deaa"`.

What is your estimation of the average age of a *Facebook* user? How many users did you visit to get this estimation?

Hint: Refer to the helper notebook `ix-lab2-helper.ipynb` *to get an example code illustrating how to query the API.* ■

[1] While the *real* Facebook does provide HTTP APIs, the one we are using for this lab is fictitious and the underlying social graph was created from scratch.

Algorithm 2.1 Random walker on *Facebook*

```
Require: source node $s$, number of nodes $N$
  $u \leftarrow s$
  $i = 0$
  while $i < N$ do
      Get the age the node $u$
      Select node $v$ uniformly at random from the neighborhood of $u$
      $u \leftarrow v$
      $i \leftarrow i + 1$
  end while
```

Exercise 2.8

A study whose title is “Facebook, by Facebook” is released, and it reveals that the average age is 45.

1. How close is your estimation to the true average age?
2. In case there is a discrepancy, how do you explain it?
3. Implement a modification of the algorithm in order to have a more accurate estimate of average age. Your solution should require neither additional seeds nor more users crawled. What is your new estimation of the average age of a *Facebook* user?

Hint: You may want to visualize the results over time to make debugging easier, and help you get some additional insights. ■

2.3 Epidemics

In the following part, we will explore how classical epidemic processes propagate over complex networks. In particular, we will simulate stochastic SIR epidemics. For this task, we will use a modified version of the New York City road network

```
nyc_augmented_network.json
```

located in the `data/` folder. This graph is augmented with geographical *meta*data and is stored as a `json` formatted file with the following schema:

```
{
    "directed": false,
    "multigraph": false,
    "links": [{"source": 0, "target": 26480}, ... ],
    "nodes": [{"id": 0, "coordinates": [-73887234, 40714842]}, ...]
}
```

In particular, `"links"` is the list of directed edges and `"nodes"` is the list of nodes. Each node has a unique `"id"` and a tuple of `"coordinates"` corresponding respectively to its longitude and latitude. We will use this *meta*data for visualization. Hint: use `networkx.readwrite.json_graph`.

A continuous-time stochastic SIR epidemic is a stochastic process spreading over a network as follows:

- All nodes are in one of three states:
  - ***Susceptible*** when they are healthy but not immune, so they can catch the disease.
  - ***Infectious*** when they are sick, so they can spread the disease to their neighbors.
  - ***Recovered*** when they are healed (or dead) and immune. They are not infectious and cannot be re-infected. Once a node in this state, it always stays in this state.

- At the beginning, all nodes are susceptible except one infectious **source** node.
- Infectious nodes **infect** each one of their neighbors with an exponentially-distributed random delay of rate β (*i.e.*, in $1/\beta$ days on average).
- Infectious nodes **recover** from the disease with an exponentially-distributed random delay of rate γ (*i.e.*, in $1/\gamma$ days on average). Recovered nodes always remain recovered.

Simulating continuous-time stochastic SIR epidemics is quite tricky. Therefore, we provide you with a Python class `SimulationSIR` in the module `epidemics_helper`. You can find more details on how to use `SimulationSIR` in the notebook `ix-lab2-helper.ipynb`.

Exercise 2.9 Simulate an epidemic outbreak

An outbreak of a new strand of the SARS-CoV-2 virus, causing the COVID-19 disease, just started to spread in New York. Experts report that the epidemic is highly infectious (with an infection rate $\beta = 10.0$) and takes quite a long time to recover (with a recovery rate $\gamma = 0.1$).

1. The first infection was observed at node `23654`. Simulate the COVID-19 epidemic with the aforementioned parameters for 100 days.
2. Plot the evolution of the epidemic over time. In particular, plot the percentage of susceptible, infected and recovered nodes over time.
3. How long until 60% of the population is infected (at the same time)? recovered?
4. Use the coordinates of the nodes to visualize the graph. Show susceptible, infected and recovered nodes in different colors to differentiate them. How does the graph look like on day 1? day 3? day 30? ■

2.3.1 Stop the apocalypse!

The World Health Organization predicts that this COVID-19 strand will become pandemic within a few weeks if we do not act now to stop it. Your task is to design an intervention strategy to prevent the apocalypse by limiting the *total* number of infected people. Some experts claim that the best intervention strategy is to block roads, *i.e.,* remove edges in the network.

Exercise 2.10 Strategy 1

The naive approach consists of removing edges at random.

- Implement this strategy with a budget of 1000 edges to remove.
- Simulate the epidemic multiple times with the same parameters, *i.e.* $\beta = 10.0$ and $\gamma = 0.1$, starting from **randomly** selected source nodes.
- Is this strategy effective?
- On average, how many people are in a healthy, infected, recovered state on day 30?
- What happens if you increase the budget (*e.g.*, to 10000 edges)? ■

Exercise 2.11 Strategy 2

Now, instead of removing edges at random, we ask you to come up with a more effective strategy. You are only allowed to remove edges in order to slow down the epidemic, but you should remove them in a more clever way using properties of the graph.

- Clearly explain your strategy.
- Implement it.
- Simulate the epidemic multiple times with the same parameters, *i.e.* $\beta = 10.0$ and $\gamma = 0.1$, starting from **randomly** selected source nodes.
- Is your strategy more effective?

In our reference implementation, we manage to keep 70% of healthy nodes on average on day 30 by removing 2500 edges. How does your strategy compare? ■

2.4 PageRank

In the previous part, you had to rank the edges of the network to prevent the spread of an epidemic. In many other applications, it is required to rank the nodes of very large networks, *e.g.* webpage ranking for a search engine.

In this part of the lab, we will implement the well-known ranking algorithm PageRank, one of the key ingredients that made the Google search engine so successful. We represent the Web as a graph where webpages (nodes) are connected through hyperlinks (directed edges). In the `data/` folder we provide you with three different "webs" on which you will run PageRank. Two of them are simple, artificial toy examples (`components.graph`, `absorbing.graph`). The last one, `wikipedia.graph`, has been extracted from a small version of the English Wikipedia. The titles of the Wikipedia pages are given in the tab-separated file `wikipedia_titles.tsv`. Each graph is stored as an adjacency list. For example, `absorbing.graph`:

```
0    1 4
1
2    3
3    0 1 2
4    1
```

2.4.1 Random surfer model

In this section we will code our first implementation of PageRank, closely following the random surfer model. In this model, a “surfer” starts by choosing a webpage uniformly at random from the list of all pages; then, she selects a hyperlink on the current page uniformly at random and continues to the next webpage. The surfer continues this process of selecting links at random from successive webpages.

Each time a page is visited, we increase this page’s counter. The PageRank score of the webpage is then defined as the normalized number of times that webpage is visited during this random surfing, i.e. it represents the probability that a random surfer happens to be on this page at any moment in time.

Exercise 2.12
Implement the random surfer model and run it on `components.graph` and `absorbing.graph`. What happens? What do you think is causing this behavior? ■

Exercise 2.13
To overcome these issues, two ideas were proposed by the inventors of PageRank (Larry Page and Sergey Brin, co-founders of the Google search engine).

1. When we reach a dangling node (i.e. a node with no outgoing link), the surfer starts from a new node chosen uniformly at random.
2. At each iteration, with small probability (called the damping factor), we start surfing from a new node chosen uniformly at random. This is called a random restart. We will use a damping factor of 0.15.

Implement this modified version of the random surfer model and run it on the two graphs `components.graph` and `absorbing.graph`. Do you think that the PageRank scores make intuitive sense? ■

2.4.2 Power iteration method

The problem with our algorithm so far is that it is very slow to converge and needs a huge number of iterations for big graphs such as `wikipedia.graph`. To overcome this limitation we will use the *power iteration method*, which we explain in this section.

Let $W(V,E)$ be the web graph, with $N = |V|$ nodes, and let o_u be the outdegree of node u (i.e. the number of outgoing links.) The transition matrix of the Markov chain describing the random walk on the web graph, H, can be written as

$$H_{u,v} = \begin{cases} 1/o_u & (u,v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

If our random surfer arrives at a dangling node, we want it to continue surfing from a new webpage chosen uniformly at random. We define $\mathbf{w}$ as the indicator vector of dangling nodes. The new transition matrix is then defined as

$$\widehat{H} = H + \frac{1}{N}(\mathbf{w} \cdot \mathbf{1}^T).$$

Graphs with absorbing nodes are not the only classes of graphs which need special care. For example, assume we have a graph with two connected components. A random surfer can walk only in one of the two components: the one where the surfing started. To solve this, we use random restarts. At every iteration, flip a coin, with probability θ walk on $\hat{H}$ and with probability $1-\theta$ jump to a new random webpage. The final transition matrix G, called the *Google matrix*, is thus defined as

$$G = \theta \widehat{H} + (1-\theta)\frac{\mathbf{1} \cdot \mathbf{1}^T}{N}. \tag{2.1}$$

Finding the PageRank is then equivalent to find the (unique) distribution π that satisfies

$$\pi^T = \pi^T G.$$

One way to solve this equation is the power iteration method. Given an initial vector $\pi^{(0)}$ (usually $1/N$ for all elements) the power iteration method successively computes the matrix multiplication operations

$$\pi^{(k)} = \pi^{(k-1)} G, \; k = 1, 2, \ldots \tag{2.2}$$

until convergence.

Exercise 2.14 Power Iteration method

1. Implement the power iteration method as described above.
2. Apply it to the `wikipedia.graph`.
3. What are the top 10 pages with the highest PageRank score? ■

2.4.3 Gaming the system

In this final part of the lab, we are interested in coming up with a strategy to increase the PageRank of a certain page by adding edges to the graph.

Exercise 2.15

Can you come up with an algorithm to maximize the page rank of the page *History of mathematics* by adding at most 300 edges.

1. What is the original PageRank score of the page?
2. Clearly explain your strategy and implement it.
3. What PageRank score do you manage to obtain? ■