Lab 5 – Routing

Assistants

Cees Portegies
Frederick Kreuk
Johannes Blaser
Kyrian Maat
Niek van Noort
Lu Zhang
Zi Long Zhu

You can contact all TAs by emailing the following address: nns18-ta@list.uva.nl

Lab date

09:00 Tuesday September 25th, 2018

Deadline

23:59 CEST Monday October 01st, 2018

Total points

10

1. Abstract

This assignment focuses on understanding how packets are routed from source to destination through a number of routers. You will learn how to manipulate an Internet topology with NetworkX and how to route between different regions using the shortest path.

2. Preparation

For this assignment, you must use the Python library NetworkX to manipulate graphs. Have a look at https://networkx.github.io/.

Using NetworkX, you will examine a dataset based on a real network topology, in this case the network infrastructure of the European Research and Educational Network of Europe (GÉANT). You can find the file for the topology, topology.gml, together with the template for the code, lab5-XXX_Y.py, in the archive lab5_files.zip on Canvas.

For the calculation of the cost in a shortest path, the Python library GeoPy is used. For more information, see https://geopy.readthedocs.io/.

3. Submission

IMPORTANT: Your code has to meet the following requirements. Failing to meet this can result in the submission **not** being graded.

1. Your code **must** be PEP8 and PEP257 compliant. It should contain docstrings and pass a PEP8 checker. See: http://pep8online.com.
2. Your code **must** be written in Python 3. Python 2 submissions **will not** be graded.
3. Your code must make use of NetworkX version 2. Have a look at https://networkx.github.io/documentation/stable/install.html and https://networkx.github.io/documentation/stable/release/migration_guide_from_1.x_to_2.0.html for instructions.

Your code will largely be graded automatically. Make sure to use the template code. Do not change the format and always comply to the return type of each function, even if you did not finish all the tasks. Points will be deducted if you fail to comply. Submit your working code in a Python script, with the following naming convention: lab5-<student_number>.py, for example lab5-7.py.

NOTE: Your code **must** run solely with the files you provide. Include any relevant files -- also including the topology.gml. If your code does not run out-of-the-box points may be deducted or your submission ignored. **DO NOT** include any unnecessary files however (folder like `_MAC_OSX`).

Please do not forget to include your name, your student number and a short description of the program at the top of the file. Write comments for your functions using docstrings. You can find an example at: https://en.wikipedia.org/wiki/Docstring#Python.

4. Assignment

This assignment is split into multiple tasks. Each task is then split into two parts. In the first part you will implement a general concept like determining the shortest path between two given nodes in a given network. The second part of the task then uses said implementation on the provided topology to answer a number of questions.

Make sure to properly explain your implementation in the docstrings and code comments. Failing this your implementation and/or answers can be ignored.

Task 0 – Graph visualization

A .gml file contains text-based information on the network which can be imported into NetworkX. This text file however doesn't provide great understanding of the structure of the topology. To visualise the topology and understand the concepts better you can visualise the provided topology easily using the example code provided at: https://networkx.github.io/documentation/stable/tutorial.html#drawing-graphs.
You do not have to hand in this code.

Task 1 – Shortest Path (2 pts)

You are (obviously) not allowed to make use of the shortest path functionality provided by NetworkX or any other library.

General tip: You are of course allowed to **test** your own code with the NetworkX built-in tools as a comparison. That should give you a good idea of how good your own code is.

In this task you will implement your own shortest-path algorithm. You are free to choose whichever shortest-path algorithm you like provided you implement it (correctly) yourself.

As a measure of distance you should use physical distance. The provided topology contains longitude and latitude coordinates. GeoPy can convert these to physical distances for you. Make sure to use **geodesic** distance calculation (see: https://en.wikipedia.org/wiki/Geodesic for more information). **You do not have to implement your own distance calculation but should use GeoPy for that.**

Tip: A node in a NetworkX graph G is characterized by its label. For example, if you would like to obtain the attributes of the Ireland (IE) node, you can retrieve it with G.nodes['IE'].

Task 1a – Generic Shortest Path Algorithm

Implement the function `task1a()` that receives a graph and the **labels** of two nodes. The function should then determine the shortest path from the **first** received node to the **second**. It should also determine the length of this path **in kilometres**. It should thus return a tuple containing:

1. A list of nodes that **consecutively** forms the shortest path from the **first** received node to the **second**.
2. The total distance of the path **in kilometres -- rounded to the nearest integer**.

Do not forget to clearly explain your implementation in the code comments.

Task 1b – Topology

Implement the function `task1b()` that uses `task1a()` to find two shortest paths:

1. From the Netherlands to Ireland
2. From the Netherlands to Switzerland

Return the **longest** shortest path from these two. Does the answer surprise you? Explain in the docstring of function `task1b()` which path you expected to be longer.

Task 2 – Longest Path (2 pts)

In this task you will use your code from **task 1** to analyse all of the shortest paths in a given graph. You will start by finding the longest shortest path in a given graph. This is the **diameter** of the graph.

Task 2a – Diameter

Implement the function `task2a()` that receives a graph and finds its diameter. It must return a tuple consisting of:

1. A list of nodes that **consecutively** form the longest shortest path in the graph.
2. The total length of this path **in kilometres -- rounded to the nearest integer**.

Task 2b – Diameter of GÉANT

Implement the function `task2b()` that takes the provided topology and finds the diameter of the GÉANT network. Return both the path as a list of nodes and the distance thereof.

Task 3 – Average Shortest Paths (2 pts)

In this task you will use a different topology to analyse the average shortest path length in a different network. The new network is the Japanese JGN2plus network. It's smaller so that complicated optimisations are not essential to having your program run in a reasonable amount of time.

Task 3a – Average Shortest Path Length

Implement the function `task3a()` that receives a graph. It must return the average shortest path length in the provided network. Be careful **not** to round the answer in this step. Obviously you **are not allowed to use NetworkX built-in functionality for calculating the average shortest path length**. You also **must** use your code from Task 1a to calculate the shortest paths.

Task 3b – JGN2plus Japan

Implement the function `task3b()` that uses your code from the previous task to calculate the average shortest path length in the **JGN2plus Japanese** topology.

Task 4 – Link Construction (2 pts)

In this task you will become the engineer responsible for extending the existing network. You must build a new link in the network. You wouldn't want to just place the link anywhere however. Instead you must determine the optimal place for a new link. To determine this you will have to find a potential new connection that would reduce the average shortest path length the most. You **must** reuse your code from the previous task.

Task 4a – Optimising Averages

Implement the function `task4a()` that receives a graph. It should look at all edges that could be added to the graph and calculate what the average shortest path length would be **with** and **without** that path. Comparing the difference will tell you how much the

average shortest path length changes. Your function should find the edge that maximises the **decrease** in average shortest path length.

Task 4b – Building New Connections

Implement the function `task4b()` that uses the **second** provided topology (the Japanese topology) to find which edge you -- as an engineer -- could best add to the existing network. It must return a tuple containing the following:

1. Node 1 of the new edge
2. Node 2 of the new edge
3. The **difference** (absolute) between the old and new average path lengths (the gain)

Task 5 – Node Destruction (2 pts)

In this task you will become the opposite -- you will break nodes. Or more specifically, you will analyse which nodes are the most central in the network. If a very central node were to fail the effects on the rest of the network would be greater than if a remote node would fail. Again, you **must** reuse code from your previous tasks were applicable. You are also **not** allowed to use built-in NetworkX (or any other library) functionality for calculating the (betweenness) centrality. Have a look at Betweenness Centrality.

Task 5a – Betweenness Centrality

Implement the function `task5a()` that receives a graph. It should calculate the betweenness centrality for all of the nodes in the graph. It should return a dictionary where the **keys** are the node IDs, and the **values** are the betweenness centrality. The values should be **normalised**.

Task 5b – Worst Case

Implement the function `task5b()` that uses the **first GÉANT** topology to find the **three** nodes that would have the greatest detrimental effect should they fail. It should return a list containing the IDs of the three most central nodes. The first should be the most central node, and thus the worst if it were to fail.