Arnav Mankad

Professor Kontothanassis

CDS DS 210

December 12, 2023

Final Project Report

Graph Description:

The graph that I am using is one where the nodes represent cities in Europe and the edges between pairs of cities represent highways that connect the two cities. The network is undirected, which means each edge is a two-way connection. Since it is a network of highways, each city is only connected to a small set of cities, though there are 1174 in all. However, traversing different paths should theoretically allow me to go between any two vertices. I, therefore, chose to pursue a project that allows me to find the distance between each vertex pair in the graph. The URL below is where I got the graph from; the website also contains important information about the dataset, including the size, format, etc.:

Eurodata.csv: http://konect.cc/networks/subelj_euroroad/

Project Overview:

My project involves calculating the distances (shortest distances) between each pair of vertices on a graph. The goal is to see which vertices/nodes are most closely related to each other; in the context of this dataset, the distances I get can tell me which cities are closer to each other (or better connected via highways). Another thing I implemented was "degrees of separation," a theory that suggests that, for the most part, any two people in the world are connected through a chain of relatives/friends/acquaintances. In the context of my project, if I were talking about 6 degrees of separation, I would be able to start at a particular city and reach almost any other city in Europe using a maximum of 6 roads. In my program, I calculate the distances from each node to all the others with degrees as a limitation. Cities that cannot be connected within a set number of degrees are filtered out. One of my functions actually calculates the percentage of connections; I hypothesized that the percentage of connected vertex pairs would increase as the number of degrees of separation increases.

Methodology:

There were two main parts to this project that have been split up into two nodes. The first step was to convert the csv file into a usable undirected graph. I did this by creating a set of functions that turned the data into an adjacency list. Each line has a vertex/node, and a list of vertices that are directly connected to it. Refer to graph.rs in my code submission to see what each individual function does.

Next, to I implemented the Breadth-First Search (BFS) algorithm to calculate the distances between nodes on the graph. The algorithm implementation involves starting at a particular vertex and checking its immediate neighbors (nodes within the adjacency list of the starting vertex). Then, it explores the next level, in that it checks the neighbors of the starting vertex's neighbors. The system continues checking neighbors at each level until it has processed every node in the graph. Each time the depth increases, the distance from the starting vertex increases by 1 (for example, a neighbor of a direct neighbor is "2" away from the starting vertex). This way, I can store the distance of every reachable node in the graph from the starting node.

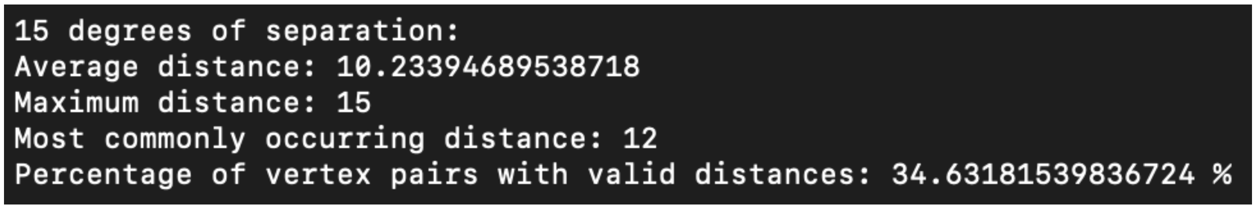
After implementing BFS, a function filters out all distances that are larger than a specific degree that the user can input (setting all distances greater than the maximum depth/degree to "None"). Then, I calculate the average distance, max distance, mode distance, and percentage of connected pairs for all valid distances (those that are not 0 or not larger than the specified degree). This allows me to analyze how the connectivity of the graph changes as the maximum distance threshold increases. Note that, the way the program has been set up, the program sets the distance of one vertex to itself to 0. That is why distances of 0 are filtered out when doing the mathematical calculations. Refer to bfs.rs for the functions associated with BFS calculations.

How to Run:

The most effective way to run the program is to go to the terminal, navigate to the directory which contains the src file for the code, and using the following command: `cargo run –release`. While `cargo run` can also be used, `cargo run –release` is faster because it uses previous calculations to make new ones. It is important to have euroroad.csv downloaded to run the program, but note that this dataset is accessible from the GitHub repository the code files are in. There is also test code in the main.rs file which allows me to verify the functionality of the average distance, max distance, mode distance, and valid distance percentage functions, using a sample graph I created. To run the test code, go to the terminal and type `cargo test`.

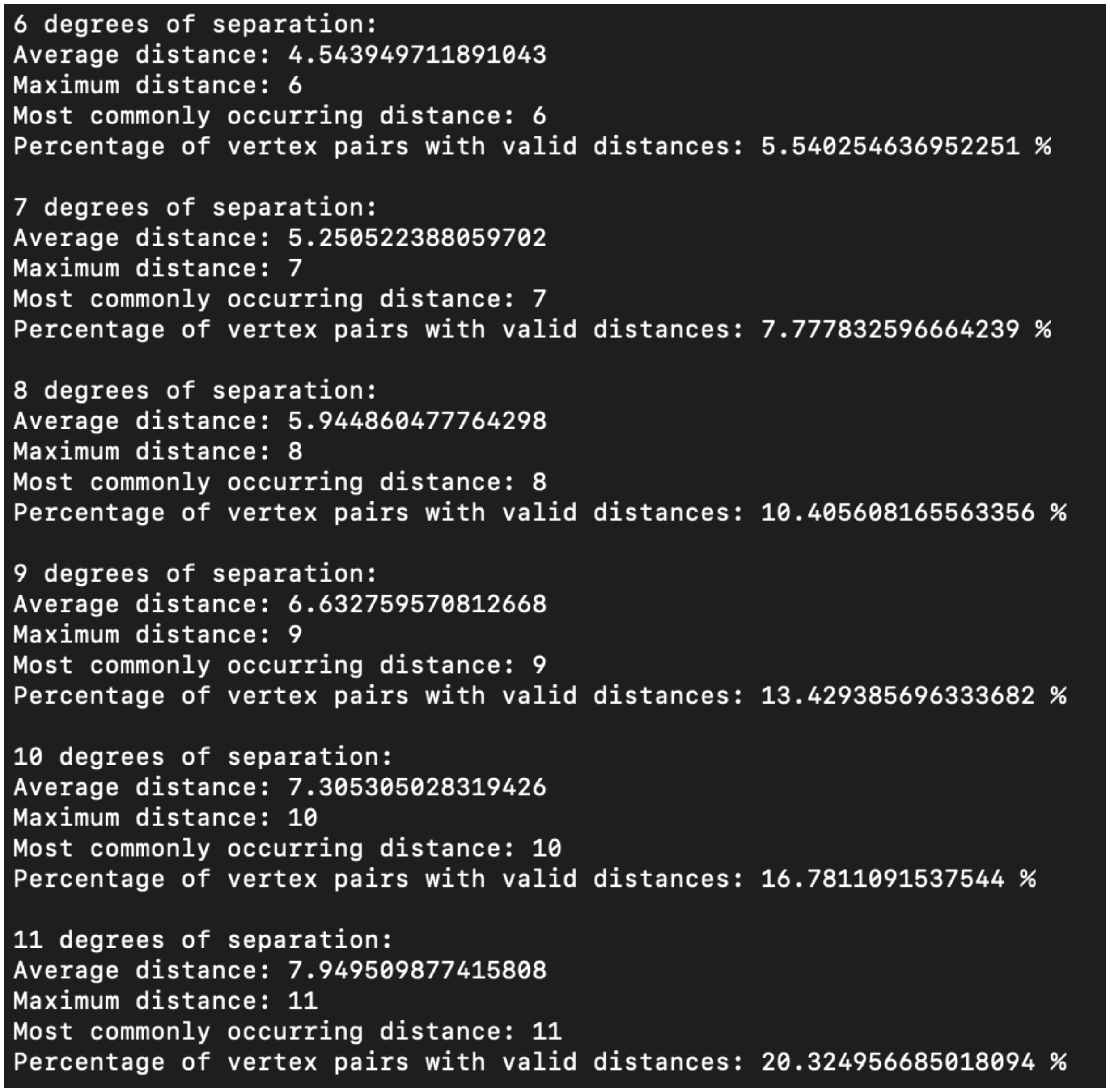
Output and Information:

Below is an example of the output when the number of degrees of separation is 15:



```
15 degrees of separation:
Average distance: 10.23394689538718
Maximum distance: 15
Most commonly occurring distance: 12
Percentage of vertex pairs with valid distances: 34.63181539836724 %
```

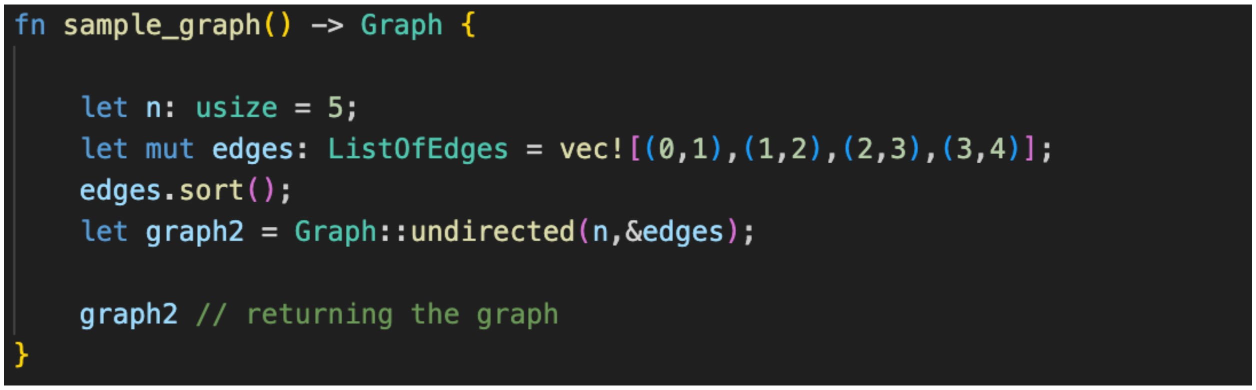
The output shows the number of separation degrees, the average distance between valid node pairs (pairs that do not have a distance larger than the max separation degree of 15), the maximum distance between nodes (it should be 15 because that is the maximum separation degree), the mode distance, and the percentage of valid vertex pairs. This percentage is the number of valid pairs divided by the total number of node pairs (multiplied by 100). It is important to note that the percentage can never reach 100% because distance between a vertex and itself is always 0 (invalid, as a result), and my program stores the distance between each vertex and itself. Refer to the image below for the output for degrees between 6 and 11. Note that my code runs for degrees 1 to 25 (the full output is not shown):



```
6 degrees of separation:
Average distance: 4.543949711891043
Maximum distance: 6
Most commonly occurring distance: 6
Percentage of vertex pairs with valid distances: 5.540254636952251 %

7 degrees of separation:
Average distance: 5.250522388059702
Maximum distance: 7
Most commonly occurring distance: 7
Percentage of vertex pairs with valid distances: 7.777832596664239 %

8 degrees of separation:
Average distance: 5.944860477764298
Maximum distance: 8
Most commonly occurring distance: 8
Percentage of vertex pairs with valid distances: 10.405608165563356 %

9 degrees of separation:
Average distance: 6.632759570812668
Maximum distance: 9
Most commonly occurring distance: 9
Percentage of vertex pairs with valid distances: 13.429385696333682 %

10 degrees of separation:
Average distance: 7.305305028319426
Maximum distance: 10
Most commonly occurring distance: 10
Percentage of vertex pairs with valid distances: 16.7811091537544 %

11 degrees of separation:
Average distance: 7.949509877415808
Maximum distance: 11
Most commonly occurring distance: 11
Percentage of vertex pairs with valid distances: 20.32495668501809 4 %
```

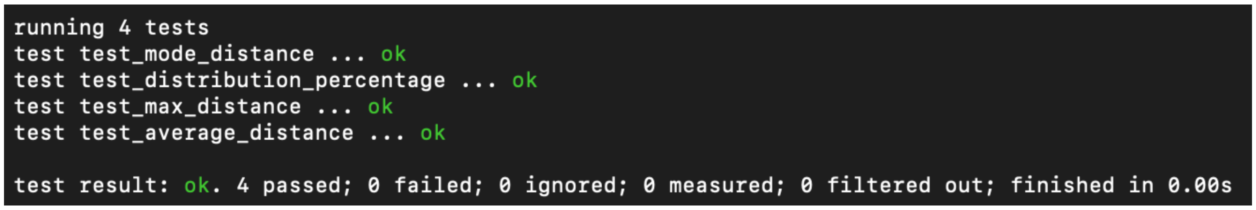
Test and Information:

In order to test my functions that calculate the average distance, max distance, mode distance, and valid vertex pair percentage for each degree, I created a sample graph using the code below:



```rust
fn sample_graph() -> Graph {

    let n: usize = 5;
    let mut edges: ListOfEdges = vec![(0,1),(1,2),(2,3),(3,4)];
    edges.sort();
    let graph2 = Graph::undirected(n,&edges);

    graph2 // returning the graph
}
```

I expect the average distance to be 1 when the degree of separation is 1. I expect the maximum distance to be 2 when the degree of separation is 2 (0 ~ 1 ~ 2, for example). I expect the mode distance to be 1 when the degree of separation is 2. Finally, I expect the percentage of valid pairs to be 80% when degree of separation is 4, since the only invalid pairs are (0~0), (1~1), (2~2), (3~3), (4~4). I used these expected values and the calculated values in my test (using assert_eq!) and found that each function works as expected:



```
running 4 tests
test test_mode_distance ... ok
test test_distribution_percentage ... ok
test test_max_distance ... ok
test test_average_distance ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Conclusions:

Through this project, I learned how well-connected the European cities represented in my graph are via the highway system. As the number of degrees of separation increased, the percentage of valid vertex pairs increased as well. I also learned that the increase in degrees of separation and the average distance does not have a linear relationship: at 6 degrees of separation, the average distance is almost 8, while the average distance is around 14.5 for 25 degrees of separation. What's more, I learned that after a certain point, increasing the number of degrees of separation does not have a tremendous impact on the data (particularly the distance calculation). The mode distance, for example, is 12 from 12 degrees of separation to 25. While there are some outliers that are less connected, it seems a majority of the cities tend to be connected within 12 degrees of separation. I learned a lot about graph theory and data analysis during this project and I am excited to explore it more in the future.

Acknowledgments:

1) I referenced Professor Kontothanassis' lecture 28 notes to learn about Breadth-First Search and key functions that I used in my program to convert a dataset into an undirected graph
2) I referenced https://crates.io/crates/csv (the link was provided in class) to create a csv reader for the euroroad data