Azalea Rohr

Professor Kontothanassis

CDS DS 210

May 1st 2024

210 Final Project Report

Project Overview

My final project is about calculating the shortest distances between each pair of vertices on a graph. The graph that I am using is one where the nodes represent cities in Europe and the edges between the pairs of cities represent highways that connect the two cities (http://konect.cc/networks/subelj_euroroad/). This network is undirected and unweighted, so each edge is a two-way connection since it's a network of highways, however, in the real world highways are “weighted” since they have different lengths. Each city (or node) is connected to a relatively small set of cities since there are only 1174 nodes in total. Traveling through different paths in this network should allow me to go between any two vertices, so within this project I am finding the distance between each vertex pair of the graph. The main goal is to see which vertices are most closely related to each other, and in the context of this data set, the distances that I calculate can tell me what cities are closer to each other, meaning they are better connected with the highways. For this project, I used the theory of “degrees of separation” which suggests that two people in the world are connected through different people whom they call relatives, friends, and acquaintances, and you can get to another person through this chain of 6 other people. In the context of highways, the degree of separation theory is that you can start at any particular city and reach another city in Europe using a maximum of 6 roads, theoretically. Within my code, I calculated the distance from each node to all of the other nodes with the number of degrees as a limitation, and cities that were not able to be connected within a determined number of degrees (edges) were filtered out. I also calculated the percentage of

cities that are connected by a certain number of roads, and I believe that the percentage of connected vertex pairs would increase as the degrees of separation increase as well.

Methodology

The first step was to make the CSV file into a readable undirected graph, by creating a set of functions that turned the data into an adjacency list, which is in the graph.rs portion of my code. Each line has a vertex and another column of vertices that are directly connected to it. I also implemented a Breath First Search algorithm to find the distances between different nodes on the graph. The implementation of the algorithm is created by starting at a particular vertex and looking at its immediate neighbors (by looking at the undirected graph that was derived from the CSV file). After that, it checks the neighbors of the starting vertices neighbors. The algorithm continues checking neighbors at continuously higher levels until every node on the graph has been processed. In each iteration, the depth increases, and the distance from the starting vertex is increased by 1. With this algorithm implementation, I can store the distance of every reachable node in the graph from the starting node. After that, I created a function that filters out all distances that are greater than a specified degree, so there is a maximum depth with each iteration showing the average distance, maximum distance, mode distance, and percentage of connected pairs for all legitimate distances for each specific degree. With this, I can see how the connectivity of the graph changes as the maximum number of nodes away from the starting node (maximum distance threshold) increases. To ensure that the program doesn't count the distance to the starting node to itself, that distance is reported as 0, so within the program, distances of 0 are filtered out when doing calculations.

How to Run Algorithm

To run the program go through the terminal, navigate through it directly, and find the src file for this project. Once you are there, either cargo run or cargo run –release can be used, with

cargo run –release having a faster run time since omits debug assertions, integer overflow checks, and debug info that is typically included with cargo run. The euroroad.csv should also be downloaded to run the program, but this is also available within the GitHub repository that the rest of the project is in. Within the main.rs file, there are also various tests to verify the functionality of average distances, maximum distance, mode distance, and legitimate distance percentages using a simple sample graph. To run these tests, type cargo tests within the terminal.

Output and Other Information

Below is a snippet of the output, from one degree of separation to six degrees of separation, however, the code goes up to 30 degrees of separation.

```
1 degrees of separation:
Average distance: 1
Maximum distance: 1
Most commonly occurring distance: 1
Percentage of vertex pairs with distances not 0 and below 1 degrees of separation: 0.20547408501635375%

2 degrees of separation:
Average distance: 1.6523447090596612
Maximum distance: 2
Most commonly occurring distance: 2
Percentage of vertex pairs with distances not 0 and below 2 degrees of separation: 0.591028212056221%

3 degrees of separation:
Average distance: 2.3680635505411005
Maximum distance: 3
Most commonly occurring distance: 3
Percentage of vertex pairs with distances not 0 and below 3 degrees of separation: 1.260415185347492%

4 degrees of separation:
Average distance: 3.103018414225147
Maximum distance: 4
Most commonly occurring distance: 4
Percentage of vertex pairs with distances not 0 and below 4 degrees of separation: 2.293154636662033%

5 degrees of separation:
Average distance: 3.829624424143047
Maximum distance: 5
Most commonly occurring distance: 5
Percentage of vertex pairs with distances not 0 and below 5 degrees of separation: 3.716817241249213%

6 degrees of separation:
Average distance: 4.543949711891043
Maximum distance: 6
Most commonly occurring distance: 6
Percentage of vertex pairs with distances not 0 and below 6 degrees of separation: 5.540254636952251%
```

The output from the code shows the number of separation degrees, the average distance between the legit vertex pairs (they do not have a distance larger than the specified degree), the maximum distance between vertices (should be the specified degree since that's the maximum separation degree), the mode distance, and the percentage of legitimate vertex pairs. The percentage that is calculated is the number of legitimate pairs divided by the total number of vertex pairs, then multiplied by 100 to convert the decimal to a percentage. This percentage will never be able to reach 100% since the distance between the starting vertex and itself will be 0, however, the program stores the distance between each vertex and itself to then be a part of the final calculation of the percentage of vertex pairs. With the maximum degree of separation being 30 in my output, the largest percentage of legitimate vertex pairs was found to be 69%.

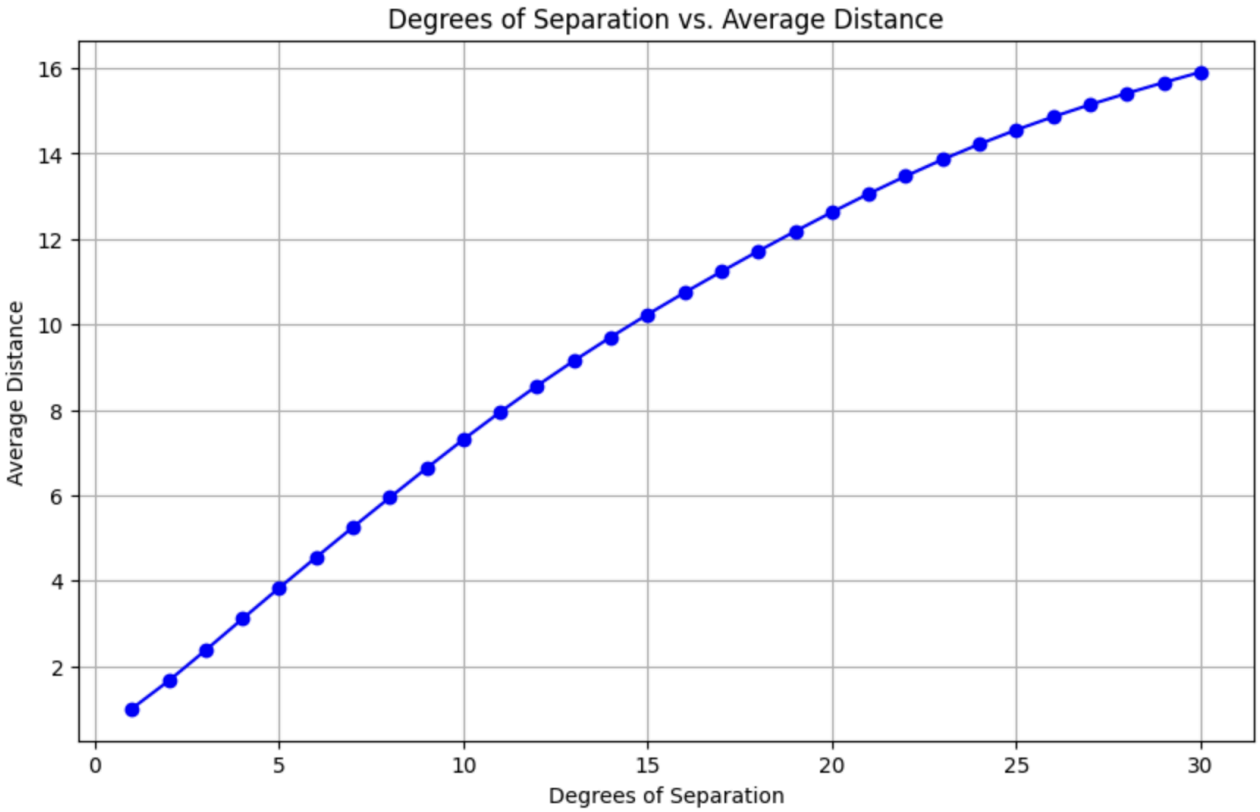
To test the functionality of my functions that calculate the average distance, max distance, mode distance, and legitimate vertex pairs for the increasing amounts of degrees of separation, I created a small sample graph to run these tests on, which is shown below.

```
fn sample_graph() -> Graph {
    let edges: Vec<(usize, usize)> = vec![(0, 1), (1, 2), (2, 3), (3, 4)];
    Graph::undirected(n: 5, &edges) // construct the graph from the edges
}
```

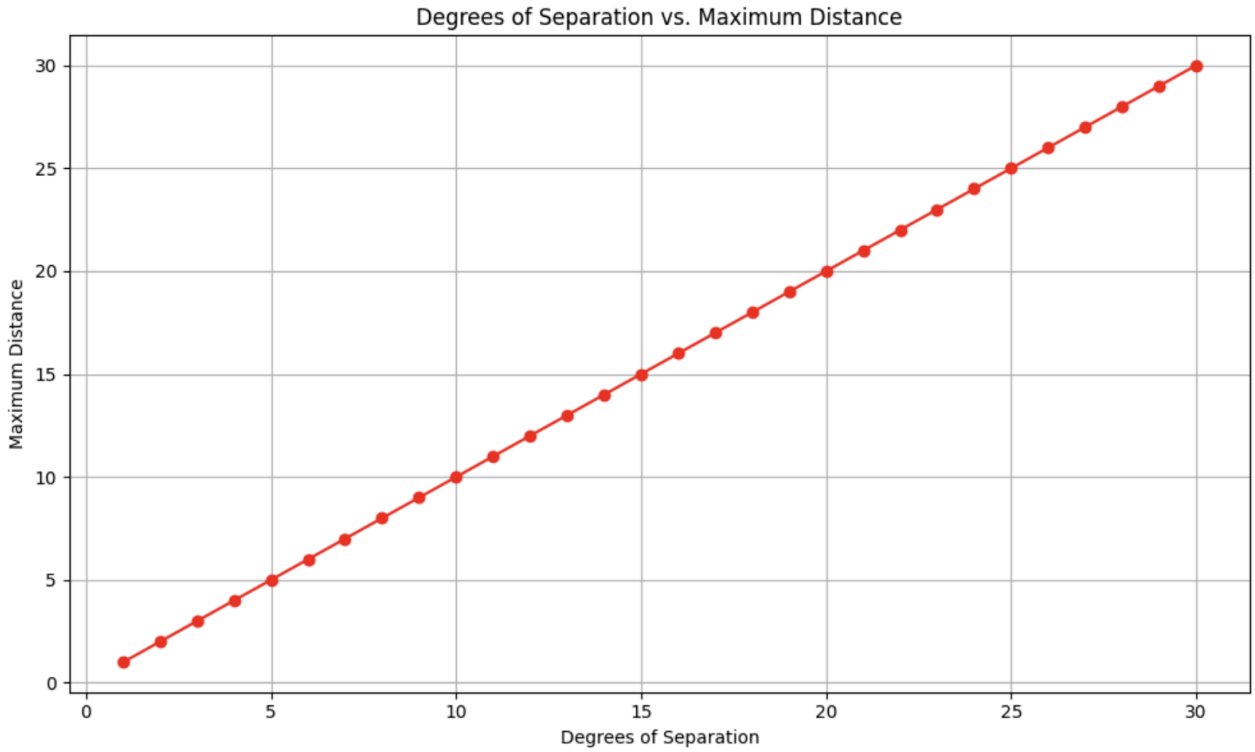
When the degree of separation is 1 I expect the average distance to also be one, which the test validated. For the maximum distance test, when the degree of separation is 2, I expect the maximum distance to be 2, which that test also passed. For the mode distance test, when the degree of separation is 2, I expected the mode distance to be 1, which the test passed. And for the distribution percentage test when the degree of separation is 4, I expected the percentage of legitimate pairs to be 80%, which the test also passed. For all 4 of my tests, using the cargo test function in the terminal, each function worked as expected and each test passed.

Analysis

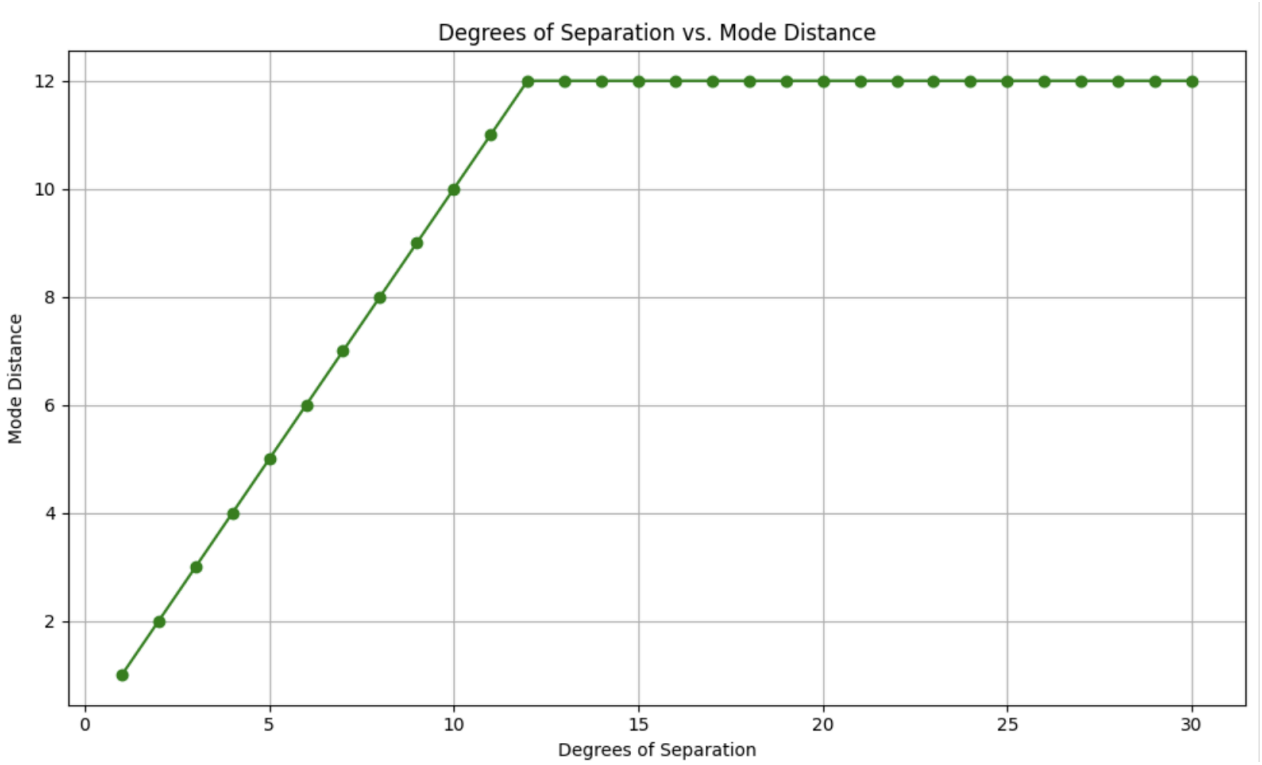
By working on this project, I discovered the connectivity of European cities through the highway system. I examined the degrees of separation between the European cities that were represented as nodes within my graph, and I discovered that the relationship between vertices doesn't follow a strictly linear relationship. Following I have plots that were created in Python using MatPlotLib of the average distance, aaximum distance, mode distance, and percentage of vertex pairs with respect to the degrees of separation to better understand these relationships.



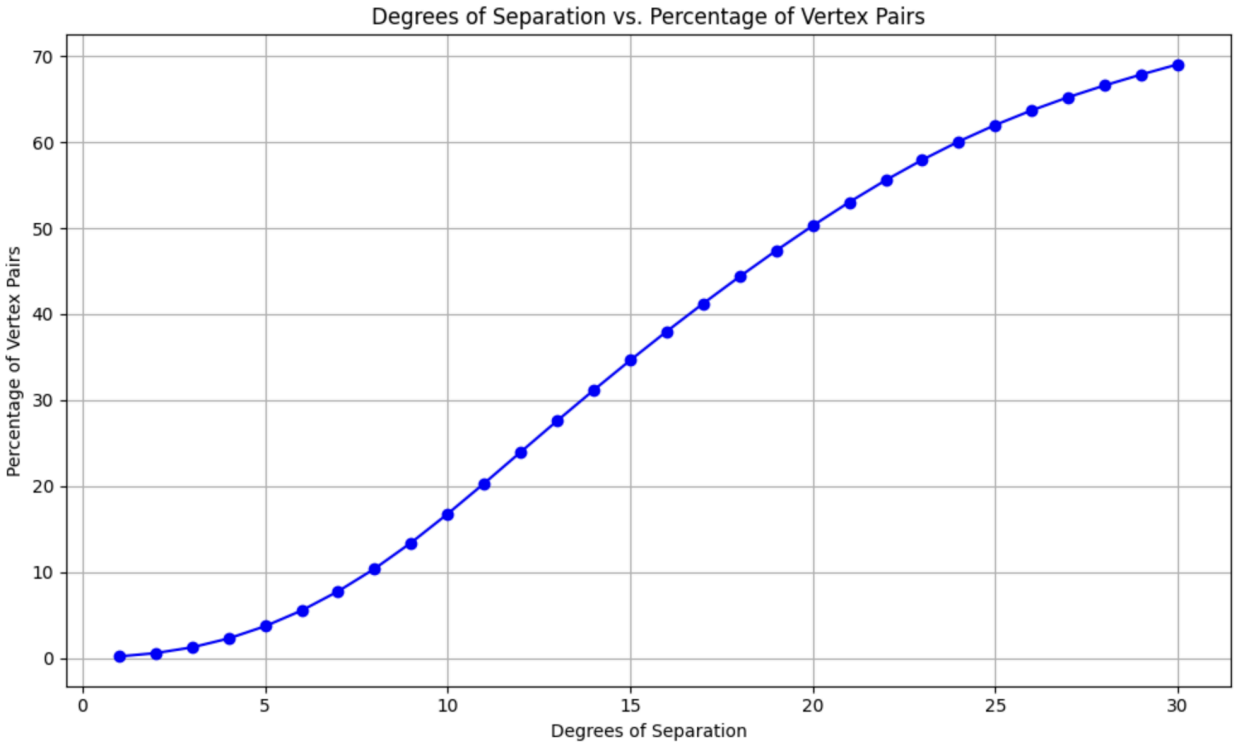
As the degree of separation increase, the average distance also increases, indicating that as the paths in the graph become longer the distance between the vertex pairs increases, which alines with general graph theory. The trend line appears to be generally linear, however it’s not exactly linear.



The maximum distance is the longest distance between any pair of vertices within the specified degree of separation. As expected, as the degree of separation increases, the maximum distance also increases at a linear rate.



One thing that I noticed is that after the maximum of 12 degrees of separation until 30 degrees of separation, the most commonly occurring distance (the mode) was 12, however, before 12 degrees of separation, the most commonly occurring distance would increase at a linear rate, showing that beyond a certain point, most cities in the network are centered around a 12-degree separation (at least for my analysis that went up to a maximum of 30 degrees of separation).



This plot shows the percentage of legitimate vertex pairs that are connected and at or below the specified degree of separation. As seen in the plot, the percentage increases as the degree of separation increases, indicating that the more vertex pairs are connected as the plot allows for a greater degree of separation. The trend on the plot is generally consistent with a steady increase from the low percentage (with one degree of separation, the percentage of legitimate vertex pairs is 0.205%) to a significant portion of vertex pairs (69% at 30 degrees of separation), showing that there is more extensive connectivity with a greater degree of separation.

Conclusion

With the findings from my project, it can also be related to real-world applications, in particular within the realm of GPS systems and urban planning. By identifying the most common degree of separation, urban planners and developers can optimize GPS routing algorithms to more accurately and efficiently guide users of the GPS through different highway networks. With my code, knowing that most cities within the Euro road CSV file are linked within 12 degrees of separation can allow GPS software to give better route suggestions that consider these common highway connections, leading to faster and more reliable travel times while driving. This analysis can also help guide infrastructure planning and traffic management. Understanding connectivity patterns in large-scale highway systems, it can help urban planners identify traffic hubs and anticipate areas where traffic congestion may occur. With this knowledge, it can help guide strategic investments from the city in expanding road capacity or GPS systems providing alternative routes, which will ultimately lead to improved traffic flow and reduced commute times. Also, the analysis of city outliers, where some cities are less connected than the majority of other cities, provided insight into regions that may need further development in terms of highways. By identifying these areas, it allows urban planners to prioritize infrastructure projects, creating more balance in terms of connectivity across the network.

According to the six degrees of separation theory, it suggests that everyone is six or fewer connections away from each other. While this theory is typically applied to human connections, it is worth noting that in the case of the European highway system, the connectivity often exceeds this boundary with the most common degrees of separation being beyond six degrees (the mode being 12 in many cases), suggesting that the connectivity in large scale transportation systems may require more degrees of separation than commonly assumed. Overall working on this project helped advance my understanding of graph therapy and its practical applications and also gave a foundation for future research. It shows the relevance of

rust coding and how undirected graphs can aid in real-world challenges, ultimately contributing to a more interconnected and accessible world.