Jin Kyu (Jaden) Cho

DS 210 Final Project Report

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

This report demonstrates the analysis of the European E-road network, focusing on its connectivity, efficiency, and key structural aspects. Utilizing the Euroroads dataset, which represents a network of intersections: cities (nodes) connected by E-roads (edges), the study delves into understanding the intricate web of roadways crisscrossing the Europe continent. The dataset is feasible with the size of 1174 nodes, and it only takes less than a second when you do cargo run or – release.

Using Rust, I tried to give detailed processing and examination of the network's structure through various metrics, including connectivity, average shortest path length, separation distribution, and separation statistics.

At first, when I looked through the “euroroad.csv”, I looked for the elements that are needed for the data preprocessing. I checked for any missing values that needed to be removed, but there were none. I also checked the range and values in the dataset to ensure they make sense for the application, but it was already set up in a reasonable range. Therefore, the only processes I had to do in the module, called data_preprocessing, were converting a csv file into a format that is easily readable in Rust and making an adjacency_list function which is useful to represent a graph like a road network by transforming a format that is more suitable for graph algorithms. Each list corresponds to a node in the graph and contains the nodes that are connected to it.

For the next module, I made degrees_separation. In this module, I have 5 functions to analyze the connectivity and separation properties of a graph. The graph is represented as an adjacency list, where each node is associated with a HashSet<usize> containing its neighbors. In fact, bfs_shortest_path_all is used as a helper function for BFS, finding the shortest path lengths from a given start node to all other nodes in the graph. In the first function, I have bfs_connectivity which computes the connectivity of each node in the graph using Breadth-First Search (BFS). For each node in the graph, it performs a BFS starting from that node and counts the number of nodes reachable from each starting node. The second function is average_shortest_path_length which calculates the average shortest path length for all pairs of vertices in the graph. The function uses bfs_shortest_path_all for each node and accumulates the total distance and the number of reachable node pairs. The average is computed by dividing the total distance by the number of pairs. The third function is separation_distribution_and_max which calculates how many node pairs have each possible degree of separation and then converts these counts into percentages. It also finds the most common degree of separation. Lastly, the function separation_statisics calculates statistical measures: mean and standard deviation of the separation distances between nodes in the graph by accumulating the sum and sum of squares of the distances obtained from bfs_shortest_path_all.

Bringing all of these modules in the main file, the results print below images.

```
Connectivity: [1, 1, 38, 38, 38, 38, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 38, 38, 38, 38, 38, 38, 38, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 10
39, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 3, 3, 3, 4, 4, 4, 1039, 1039, 1039, 1039, 1039, 103
9, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039
, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 8, 8, 8, 8, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1
039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 5, 5, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 10
39, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 5, 5, 5, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1
039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 10
39, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 103
9, 15, 15, 15, 38, 38, 38, 38, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 15, 15, 15, 15, 38, 38, 38, 38, 38, 38, 38, 38, 38, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 10
39, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 103
9, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039
, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1
039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 8, 8, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 15, 15, 15, 15, 15, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 38, 38, 38, 38, 38, 1039, 1039, 1039, 1039, 1039,
 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 3, 3, 3, 1039, 1039, 1039, 1039, 4, 4, 4, 4, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 103
9, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039
, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 5, 5, 5, 5, 5, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039
, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 15, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 15, 38, 38, 38, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 103
9, 1039, 1039, 1039, 1039, 38, 38, 38, 38, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 38, 1039, 1039, 1039, 1039, 1039, 1039, 38, 1039, 1039, 1039, 1039, 1039, 1039
, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039,
 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 4, 4, 4, 4, 1039, 1039, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 4, 4, 4, 4, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1
039, 1039, 1039, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 15, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039
, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 2, 2,
 1039, 1039, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 7, 7, 7, 7, 7, 7, 7, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 2, 2, 1039, 103
9, 1039, 1039, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 2, 2, 1039, 1039, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1039, 1039, 1039, 1039,
1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1
039, 4, 1039, 1039, 2, 2, 1039, 1039, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 8, 8, 2, 2, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1039, 1]
---------------
Average Shortest Path Length: 18.35221185263808
---------------
Separation Distribution (up to 63 degrees): [0.0, 0.4110080444295339, 0.7717656353705099, 1.3399152713451872, 2.0672397542084755, 2.8497525963944574, 3.6499838065735144, 4.478971056610185 5, 5.2
60031573550833, 6.052710692454154, 6.709161703345141, 7.093737428309595, 7.291544126724092, 7.270340178142215, 7.1341120701298815, 6.942114672696721, 6.730075186877951, 6.496541287428237, 6.274
9164549902625, 6.0402206953442805, 5.785192382263623, 5.482237336086942, 5.120317884949698, 4.7200570473356365, 4.290459239765827, 3.830653067093069, 3.412673861485932, 3.058306501624426, 2.750
704014662676, 2.5264649967830994, 2.357414338226217, 2.2113104185456125, 2.069273009551943, 1.9286879258036078, 1.7695130789149969, 1.5879724232482415, 1.3956845607660144, 1.2304099478470005, 1
.0834346330191953, 0.9312309473081878, 0.7964551645411887, 0.6933400721224717, 0.5925487000962891, 0.4955333737079752, 0.42233618134313944, 0.3549482899596399, 0.28407481798733863, 0.2210439023
398412, 0.17108391390035016, 0.12925694683472974, 0.09759625648644762, 0.07029254187416764, 0.050831383586691474, 0.03834138647681871, 0.028175109759480416, 0.020913483532810206, 0.014523252453
340421, 0.009294881570137869, 0.0060997660304029765, 0.0037760456378685093, 0.002323720392534467, 0.0011618601962672336, 0.0002904650490668084]
---------------
Degree of Separation with the maximum percentage: 12
Maximum percentage of valid connections: 7.291544126724092
Mean of Separations: 18.35221185263808
Standard Deviation of Separations: 9.429788469361348
(base) chojinkyu@jojingyuui-MacBookAir final_project % 
```

The connectivity revealed a significant diversity in the connection density among its nodes. Nodes with a connectivity value of 1 are likely isolated endpoints or weakly connected nodes, while nodes with high connectivity values, such as 1039, suggest they are major junctions, indicating that most nodes are part of a large interconnected component. This is further evidenced by the average shortest path length computed to be 18.35, suggesting that, on average, about 18 roads are needed to travel from one node to any other node in the network. This value reflects a moderate level of efficiency, indicating the number of steps necessary to connect two arbitrary locations. The network is moderately dense, with the most common paths being of intermediate length (around 6 degrees). The presence of node pairs at higher degrees of separation (e.g., 10, 11, 12) suggests a wide spread in the network, where some nodes are quite distant from each other. Particularly, the maximum percentage of separation is occurring at 12 degrees, accounting for nearly 7.2915% of all node pairs. This prominent spike suggests that, despite the network's moderate average path length, there is a notable tendency for node pairs to be 12 steps apart, more than any other specific path length within the network. This specific mode might indicate a structural or functional purpose, potentially highlighting an organizational pattern or how roads operate within the network's topology. Furthermore, the mean of separations, which matches the average shortest path length at 18.35, reinforces this interpretation of the network's efficiency. Additionally, the standard deviation of 9.43 indicates a significant variation in path lengths between different node pairs, pointing to a high variability in the network's connectivity. This variability reflects the diverse nature of the network, with some paths being much shorter or longer than others, a critical factor in understanding the network's overall structure and functionality.

The most interesting parts throughout the project are the discovery of the maximum percentage of separation at 12 degrees, accounting for nearly 7.2915% of all node pairs within the European E-road network, and the dataset is rightly skewed, meaning there are more pairs of nodes with shorter paths between them, and there are fewer node pairs with longer paths between them. Most node pairs can be reached through a small number of steps. These results intrigue me because in a well connected network, most node pairs are expected to be separated with a relatively small number of steps according to the six degrees of separation. I assume that this happened due to the geographical barriers, political borders, economic zones, and different urban plannings in Europe influenced how cities are connected.

Resources: The Rust Programming Language , Rand Core, Lecture Notes, TAs, ChatGPT

https://doc.rust-lang.org/stable/book/ch11-03-test-organization.html

https://rust-random.github.io/rand/rand_core/index.html

You

How can I find the connectivity in BFS

ChatGPT

To find the connectivity in a graph using Breadth-First Search (BFS), you essentially want to determine which nodes are reachable from each starting node. The connectivity for each node is the count of other nodes that can be visited starting from that node. The BFS algorithm is well-suited for this because it explores neighbors level by level, ensuring that all nodes reachable from the starting node are visited.

Here is a step-by-step description of how you could implement BFS to find connectivity:

1. **Initialization**: For each node in the graph, you initialize a visited array to keep track of which nodes have been visited. This array will prevent the algorithm from revisiting the same node multiple times.
2. **Queue**: You use a queue to keep track of the nodes to visit next. You start by enqueuing the starting node.
3. **Exploration**: While the queue is not empty, you dequeue a node and mark it as visited. For every unvisited neighbor of this node, you mark the neighbor as visited and enqueue it.
4. **Counting**: After visiting all possible nodes from the starting node, you count the number of nodes that were marked as visited. This count (minus one to exclude the starting node itself) gives you the connectivity for the starting node.
5. **Repeat**: You repeat this process for each node in the graph to determine its

starting node itself) gives you the connectivity for the starting node.

5. **Repeat**: You repeat this process for each node in the graph to determine its connectivity.

In Rust, you can use a `VecDeque` for the queue and a `Vec` to represent the visited array. Here's a simplified code snippet that demonstrates this process:

You

what does the vecdeque do and what queues can I use?

ChatGPT

`VecDeque` is a data structure in Rust's standard library that represents a double-ended queue, which allows for efficient insertion and removal of elements from both ends of the queue. In the context of the BFS algorithm, it is used like a traditional queue, where nodes are added ("enqueued") to the back and removed ("dequeued") from the front.

Here's a breakdown of how `VecDeque` is used in BFS:

- **Initialization**: A `VecDeque` is created to hold the nodes that need to be explored.
- **Enqueue**: When a node is first discovered, it is added to the back of the queue using `push_back()`.
- **Dequeue**: When it's time to explore a node, it is removed from the front of the queue using `pop_front()`.

This structure is particularly well-suited for BFS because it allows for the FIFO (first-in-first-out) behavior that is needed when exploring a graph level by level.

Regarding other queues you can use in Rust for BFS:

1. **VecDeque (recommended)**: As mentioned, `VecDeque` is ideal for BFS due to its efficient push and pop operations on both ends.

2. **Vec**: While you can use a `Vec` with `push` and `remove(0)` operations to simulate a

2. **Vec**: While you can use a `Vec` with `push` and `remove(0)` operations to simulate a queue, this is not efficient for large graphs because removing the first element of a `Vec` requires shifting all other elements, which is an O(n) operation.
3. **LinkedList**: Rust's `LinkedList` could also be used as a queue with `push_back()` and `pop_front()` methods. However, `LinkedList` is generally less cache-friendly than `VecDeque` and might perform worse for BFS because of potential overheads in memory allocation and pointer dereferencing.
4. **Custom Queues**: You can implement your own queue using arrays or other data structures, but this is typically unnecessary unless you have very specific performance requirements that the standard library structures can't meet.

In most cases, `VecDeque` is the preferred queue for BFS in Rust due to its performance and ease of use.