CRNSA

Xiang Fu

May 5, 2023

Introduction
Dataset
Getting Started
Project Writeup
Project Modules
Results
    Assortativity Coefficient
    PageRank
    Network Density
    Clustering Coefficient
    Degree Distribution
    Number of Bridge Edges
    Number of Connected Components
    Leiden Community Detection
    Core Periphery Node Detection
    Shortest Path with Landmarks
AlgoRepo
    Label Propagation Algorithm (LPA.rs)
    K-Core Decomposition (kcore.rs)
    Node2Vec Graph Embeddings (graph_embeddings.rs)
Limitations
Future Work
Pull Requests & Issues
    Submitting Pull Requests
    Reporting Issues
License
Acknowledgements

Introduction

California Road Network Schaubild Analytica (CRNSA) is a comprehensive network analysis project focused on the road network in California. The project aims to provide valuable insights into the road network's structural properties, connectivity, and traffic patterns. By analyzing the California road network, CRNSA can help urban planners, transportation engineers, and policymakers make more informed decisions about infrastructure development and traffic management.

The CRNSA project utilizes various network analysis techniques and algorithms to study the road network. These techniques include:

- Assortativity: By examining the assortativity of the CRN, we will determine the tendency of nodes with similar properties to connect with each other, which could indicate the presence of tightly-knit communities or hierarchical structures within the network.
- PageRank: This metric will help us identify the most significant nodes in the network, based on their connectivity patterns. This information can inform transportation planners about high-traffic areas that may require additional infrastructure or resources.
- Network Density: By calculating the network's density, we will assess the overall connectedness of the CRN, which could provide insights into the efficiency and potential redundancy of the network.
- Clustering Coefficient: This measure will reveal the degree to which nodes within the CRN tend to form tightly connected clusters, which can inform us about the network's resilience and redundancy.
- Degree Distribution: By analyzing the degree distribution of the CRN, we can identify the network's underlying patterns and gain insights into potential bottlenecks or critical nodes that warrant further attention.
- Connected Components: Investigating the connected components of the CRN will help us understand its overall connectivity and identify potential areas of concern, such as isolated road segments or disconnected subnetworks.
- Bridge Edges: Identifying bridge edges will enable us to pinpoint critical connections in the network that, if disrupted, could significantly impact the overall connectivity of the CRN.

- Core-Periphery Analysis: This analysis will help us understand the hierarchical structure of the CRN, identifying central nodes and peripheral regions, which can inform resource allocation and infrastructure planning decisions.
- Leiden Community Detection: By employing the Leiden algorithm, we aim to uncover communities within the CRN, which can provide insights into regional patterns, relationships, and potential areas for targeted interventions.
- Eigenvector Centrality: This metric will help us identify influential nodes within the CRN, based on their connectivity to other highly connected nodes, which can inform transportation planners about key hubs in the network.
- Shortest Path with Landmarks: By calculating the shortest paths between landmarks in the CRN, we can assess the efficiency of the network in connecting essential points of interest and identify potential areas for improvement.

Dataset

We are using California Road Network from Stanford Network Analysis Project, which is a road network of California consisting of intersections and endpoints represented by nodes and the roads connecting these intersections or road endpoints represented by undirected edges. The dataset contains 1,965,206 nodes and 2,766,607 edges. The statistics for the largest weakly connected component (WCC) and strongly connected component (SCC) are shown below:

Property	Value
Nodes in largest WCC	1,957,027
Edges in largest WCC	2,760,388
Nodes in largest SCC	1,957,027
Edges in largest SCC	2,760,388
Average clustering coefficient	0.0464
Number of triangles	120,676
Fraction of closed triangles	0.02097
Diameter (longest shortest path)	849

90-percentile effective diameter	500

The largest WCC and SCC represent a significant proportion of the road network, with over 99% of nodes and edges contained within them. The average clustering coefficient is 0.0464, indicating that the road network is moderately clustered. The number of triangles in the network is 120,676, and the fraction of closed triangles is 0.02097, suggesting that the road network has a significant number of interconnected clusters. The diameter of the road network, which is the longest shortest path between any two nodes, is 849, while the 90th percentile effective diameter is 500, indicating that the network has a relatively short average path length. These statistics can be used to guide the analysis and identify areas of the network that may be of particular interest or importance.

Getting Started

To get started with the project, follow these steps:

1. **Install Rust**: Before running the provided Rust code, you need to make sure you have Rust installed on your system. You can follow the installation instructions on the official Rust website.
2. **Clone the repository**: Clone the project repository by running the following command in your terminal or command prompt:

   ```
   git clone https://github.com/SuzzukiW/CRNSA.git
   ```

3. **Navigate to the project directory**: Use the `cd` command to change to the project directory:

   ```
   cd CRNSA
   ```

4. **Compile and run the project**: You can compile and run the project using one of the following commands:
   - For a release build (optimized for performance):

```
cargo run --release
```

- For a debug build (faster compilation, but slower execution):

```
cargo run
```

5. **Explore the code**: Now that you have the project up and running, you can start exploring the code, understanding the project structure, and making any desired changes or additions.
6. **Contribute**: If you'd like to contribute to the project, please refer to the Pull Requests & Issues section for guidelines on submitting pull requests and reporting issues.

Happy coding! 🚀

Project Writeup

If you're interested in a PDF version of our project writeup, feel free to check out the project_writeup directory. Enjoy reading!

Project Modules

- `main.rs`
- `assort.rs`
  - Calculate the assortativity coefficient of a graph. Assortativity is a measure of how similar nodes are connected in a graph. A positive assortativity coefficient indicates that nodes with similar degrees tend to be connected, while a negative value indicates that nodes with dissimilar degrees are more likely to be connected.
  - `calculate_assortativity_coefficient`: Take a reference to a Graph object and calculates its assortativity coefficient. It iterates through all the edges in the graph, and for each edge, it computes the degrees of the source and target

nodes. The function then calculates three sums (sum1, sum2, and sum3) based on the source and target node degrees. Finally, it computes the assortativity coefficient using these sums and returns the result.

- `bed.rs`
  - Responsible for finding bridge edges in an undirected graph. Bridge edges are edges whose removal increases the number of connected components in the graph. The code uses the petgraph library for working with graphs, depth-first search (DFS) traversal for exploring the graph, and the HashSet data structure for storing bridge edges.
  - `find_bridge_edges`: Takes a reference to a petgraph Graph object with undirected edges, where nodes represent points in 2D space and edges have associated weights. The function finds the bridge edges in the graph and returns a HashSet containing pairs of NodeIndex values representing these bridge edges. The function initializes a DFS traversal starting from the first node in the graph, as well as vectors to store low-link values, discovery times, and parent nodes for each node. It then performs the DFS traversal, updating low-link values, discovery times, and parent nodes, and identifies bridge edges based on these values.
  - `bridge_dfs`: A helper function that performs the DFS traversal and updates low-link values, discovery times, and parent nodes. It takes the input graph, the starting node for the DFS traversal, mutable references to vectors storing low-link values, discovery times, and parent nodes, a mutable reference to the time counter for discovery times, and a mutable reference to the HashSet for storing identified bridge edges. It initializes a stack for the DFS traversal, explores the graph using the stack, and updates low-link values, discovery times, and parent nodes accordingly. Bridge edges are identified when the low-link value of a node is greater than the discovery time of its parent.
- `cca.rs`
  - Responsible for analyzing connected components in an undirected graph. Connected components are subgraphs in which all nodes are reachable from any other node within the subgraph. The code uses the petgraph library for working with graphs, the connected_components function from the petgraph

crate for identifying connected components, and the HashMap data structure for storing nodes belonging to each connected component.

  - `analyze_connected_components`: Takes a reference to a petgraph Graph object with undirected edges, where nodes represent points in 2D space and edges have associated weights. The function calculates the number of connected components in the graph and groups nodes belonging to each connected component. It initializes a HashMap to store nodes belonging to each connected component, where the key is the component ID and the value is a vector of NodeIndex values representing the nodes in the component. It then iterates over all nodes in the graph, adding them to the corresponding connected component in the HashMap. Finally, the function converts the HashMap into a sorted vector of vectors representing the connected components, and returns the number of connected components and the sorted vector of connected components.

- `centrality_analysis.rs`
  - Contain functionality for analyzing the degree centrality of an undirected graph. It imports the necessary modules and functions, including the degree_centrality function from the centrality.rs module.
  - `analyze_centrality`: Take a reference to a petgraph UnGraphMap object and calculates its degree centrality using the degree_centrality function from the centrality.rs module. The result, which is a HashMap containing the degree centrality of each node, is then printed to the console.

- `centrality.rs`
  - Responsible for calculating the degree centrality of nodes in an undirected graph. Degree centrality is a measure of a node's importance within a network based on the number of connections it has. The code uses the petgraph library for working with graphs and the HashMap data structure for storing degree centrality values.
  - `degree_centrality`: Take a reference to a petgraph UnGraphMap object as input and calculates the degree centrality for each node in the graph. It creates an empty HashMap to store the degree centrality values and iterates over all nodes in the graph. For each node, the function counts the number of neighbors

and inserts the node index and its degree centrality into the HashMap. Finally, it returns the degree centrality HashMap.

- `cpa.rs`
  - Responsible for performing core-periphery analysis on an undirected graph. Core-periphery analysis identifies nodes in a network that are either central (core) or peripheral based on a specified degree threshold. Nodes with a degree greater than or equal to the threshold are considered core nodes, while nodes with a degree lower than the threshold are considered periphery nodes. The code uses the petgraph library for working with graphs and the HashSet data structure for storing core and periphery nodes.
  - `core_periphery_analysis`: Takes a reference to a petgraph Graph object with undirected edges and a degree threshold as input. It calculates the core and periphery nodes in the graph based on the given threshold. The function initializes two empty HashSets to store the core and periphery nodes, then iterates over all nodes in the graph. For each node, the function calculates its degree and classifies it as a core node if its degree is greater than or equal to the threshold, otherwise classifying it as a periphery node. Finally, it returns the core and periphery HashSets.
- `data.rs`
  - Responsible for reading, preprocessing, and analyzing road network data. The file data.rs contains functions to read data from a file and represent it as an undirected graph using the petgraph library. It also includes a function to compute network properties, specifically the number of weakly and strongly connected components.
  - `read_and_preprocess_data`: Take a file path as input, reads the data from the file, and constructs a RoadNetwork graph. If the file is a gzip compressed file, it uses the GzDecoder to read the file. It reads each line from the input file, ignores lines starting with '#', and processes the remaining lines by adding an edge between the nodes with a weight of 1.0.
- `leiden.rs`
  - An implementation of the Leiden algorithm for detecting communities in graphs. The Leiden algorithm is a refinement of the Louvain algorithm and is known for

its improved performance in terms of quality, speed, and stability. The algorithm iteratively refines the community structure by locally moving nodes between communities to maximize modularity, and then aggregates communities into a new graph to repeat the process.

- `leiden_communities`: This function takes a reference to an undirected Graph object and detects communities in it using the Leiden algorithm. It returns a HashMap mapping node indices to their community assignments.
- `initial_community_assignments`: This function initializes community assignments by assigning each node to its own community.
- `local_moving`: This function performs the local moving phase of the Leiden algorithm, in which nodes are moved between communities to maximize the modularity. The function updates the community assignments in place.
- `modularity_delta`: This function computes the change in modularity resulting from moving a node from one community to another.
- `refinement`: This function aggregates the communities into a new graph and updates the community assignments accordingly.

- `network_analysis.rs`
  - Functions for analyzing various properties of an undirected graph represented by the Graph type from the petgraph crate. These properties include degree distribution, clustering coefficient, and network density.
  - `degree_distribution`: This function takes a reference to a Graph object and calculates the degree distribution of the graph. It first calculates the degree of each node in the graph, and then creates an Array1 (a 1-dimensional array from the ndarray crate) to store the distribution. The function iterates through the degrees of all nodes and increments the corresponding index in the distribution array. Finally, it returns the degree distribution as an Array1

    .
  - `clustering_coefficient`: This function calculates the average clustering coefficient of a given graph. It takes a reference to a Graph object as input and iterates through all nodes in the graph. For each node, the function calculates its clustering coefficient, which is the proportion of connected pairs among its neighbors. The function then sums up the clustering coefficients of all nodes

and divides the sum by the total number of nodes in the graph to compute the average clustering coefficient. The result is returned as a f64 value.

    - `network_density`: This function calculates the network density of a given graph, which is a measure of how densely connected the graph is. It takes a reference to a Graph object as input and calculates the network density as the ratio of the actual number of edges in the graph to the maximum possible number of edges. The result is returned as a f64 value.

- `pagerank.rs`
    - An implementation of the PageRank algorithm, a widely-used algorithm for ranking nodes in a graph based on their importance.
    - `pagerank function`: Calculate the PageRank scores for all nodes in the given undirected graph. It takes a reference to a Graph object, a damping_factor (which is typically set to 0.85), and the number of iterations to perform. The function initializes the ranks vector with equal values for all nodes and then iterates through the PageRank algorithm. At each iteration, it updates the ranks vector based on the neighboring nodes' contributions and the damping factor. After completing the specified number of iterations, the function returns a sorted vector of tuples containing the node index and its corresponding PageRank score.

- `shortest_path.rs`
    - A landmark-based approach to compute approximate shortest paths in a graph. The module uses the petgraph crate for graph representation and algorithms such as Dijkstra's algorithm and Breadth-First Search (BFS).
    - `select_landmarks`: This function takes a reference to an undirected Graph object and an integer k representing the number of landmarks to select. It randomly selects k landmarks and returns a Vec of their NodeIndex values.
    - `precompute_landmark_distances`: This function takes a reference to an undirected Graph object and a slice of landmarks. It precomputes the distances from each landmark to all other nodes using Dijkstra's algorithm and returns a HashMap mapping each landmark to a HashMap of distances.
    - `approximate_shortest_path`: This function takes the start and end nodes and the precomputed landmark distances, and returns an approximation of the shortest

path distance between the two nodes using the triangle inequality.

- `find_shortest_path`: This function takes a reference to an undirected Graph object, start and end nodes, and the precomputed landmark distances, and returns the shortest path between the two nodes along with its weight. The function performs bidirectional BFS and computes the actual shortest path using the approximate distance as a guide.

Results

Assortativity Coefficient

Assortativity Coefficient: 0.007451801430078585

The assortativity coefficient is a measure that indicates the tendency of nodes to connect with other nodes that have similar properties, in this case, similar degrees (i.e., the number of connections a node has). The coefficient ranges from -1 to 1, where a value of 1 indicates perfect assortativity (nodes with similar degrees tend to connect with each other), a value of -1 indicates perfect disassortativity (nodes with similar degrees tend to avoid each other), and a value of 0 suggests that there is no correlation between the degrees of connected nodes.

In our context, an assortativity coefficient of 0.007451801430078585 indicates a **weakly positive assortative mixing pattern**. This means that there is a **slight tendency for intersections or road segments with a similar number of connections to be connected to each other**. However, given that the assortativity coefficient is close to 0, the overall tendency is relatively weak, and the network exhibits a **nearly random pattern of connectivity**.

A weakly assortative mixing pattern in the road network could have several implications: - Resilience: The road network might be relatively resilient to failures or disruptions since there is no strong preference for highly connected nodes to be connected with

each other. This means that the network is less likely to experience large-scale failures when a few highly connected nodes are removed or become non-functional. - Traffic distribution: A weak assortativity might suggest that traffic is more evenly distributed across the network, preventing the formation of significant bottlenecks or congestion in specific areas. However, it is essential to consider other factors, such as traffic demand and road capacity, to understand the actual traffic patterns. - Routing and navigation: The weak assortativity in the network might also suggest that there is a good balance between local and long-distance connections, which can facilitate efficient routing and navigation. However, further analysis on the actual travel times and the quality of the road infrastructure would be necessary to validate this claim.

PageRank

The PageRank algorithm is a widely-used method for ranking nodes in a graph based on their importance or prominence within the network. In the context of the California Road Network, it can help identify critical intersections or road segments that play a significant role in the overall connectivity and flow of the network.

The top 10 nodes with the highest PageRank scores in the California Road Network are as follows:

Node Index	Value
234679	0.0000015108195426166147
267076	0.0000014963223508171434
571924	0.0000014603242249559171
200299	0.0000014399110747491118
529105	0.0000013736346047116763
1368330	0.0000012941237629725948
751879	0.0000012704791770856644
245979	0.0000012601173654198325
1023770	0.0000012598643287972244
1686555	0.0000012570933378460 4

These results suggest that these nodes **represent significant intersections or road segments within the California Road Network**. They likely play a crucial role in the overall connectivity of the network and may serve as critical junctions for the flow of traffic. This information can help identify areas that may require special attention, such as traffic signal coordination, infrastructure improvements, or congestion mitigation strategies.

Network Density

Network density: 1.4327206657979522e-6

Network density is a measure of how connected a network is relative to the total possible connections between the nodes.

This value is very low, which is expected for a large-scale real-world network like the California Road Network. A low network density indicates that **only a small fraction of the total possible connections between nodes actually exist**. In practical terms, this means that the road network is sparse, with most roads connecting only to a few other roads.

A low network density is typical for transportation networks, as it would not be efficient or feasible to have every road directly connected to every other road. Instead, the road network is organized in a hierarchical manner, with primary roads (such as highways and major arterial roads) connecting to secondary roads (such as local streets and collector roads), which in turn connect to tertiary roads and so on.

From a transportation planning and traffic management perspective, the low network density highlights the importance of efficient routing and the need to maintain the critical connections that do exist in the network. It also underscores the potential value of improving connectivity in targeted areas, as even small improvements in the overall network connectivity can have significant effects on travel times and congestion.

Clustering Coefficient

Clustering coefficient: 0.046370270074755485

The clustering coefficient is a measure of the extent to which nodes in a network tend to cluster together. It quantifies the likelihood that the neighbors of a node are also connected to each other, forming a tightly-knit group or “cluster.” The clustering coefficient can range from 0 (no clustering) to 1 (perfect clustering).

The value we calculated out indicates **a relatively low level of clustering in the network**. In other words, while there are some areas where roads are interconnected and form clusters, the overall network is **not characterized by a high degree of clustering**.

This observation is **consistent with the structure of a typical road network**. In a road network, clustering often occurs at the local level, such as within neighborhoods or city blocks, where multiple streets intersect and form a grid-like pattern. However, as we move to larger scales (e.g., between cities or regions), the level of clustering decreases, as roads are more likely to connect distant locations rather than forming local clusters.

The low clustering coefficient of the California Road Network suggests that while there are localized areas of high connectivity, the network as a whole is characterized by a more distributed and hierarchical structure. This is expected in a large and geographically diverse state like California, where roads must connect a wide range of urban, suburban, and rural areas.

Degree Distribution

The degree distribution of a network is a histogram that shows the frequency of each degree (i.e., the number of connections or edges) that nodes in the network have. It provides insights into the connectivity patterns of the network and can reveal important characteristics of the network’s structure.

In our case, the degree distribution is as follows:

Degree	Number of Nodes
0	0
1	321,027
2	204,754
3	971,276
4	454,208
5	11,847
6	1,917
7	143
8	30
9	1
10	2
11	0
12	1

The degree distribution indicates that the majority of nodes in the California Road Network have a degree of 3 or 4, which means that **most intersections in the network are 3-way or 4-way intersection**s. This is a common pattern in road networks, **where T-junctions (3-way) and crossroads (4-way) are prevalent**.

The distribution also shows that there are a significant number of nodes with a degree of 1 or 2, which correspond to dead-end streets (cul-de-sacs) and simple bends or curves in the road, respectively.

As we move to higher degrees, the frequency of nodes decreases sharply. There are relatively few nodes with degrees greater than 5, and very few nodes with degrees greater than 8. This suggests that complex intersections with many connecting roads are rare in the network.

Number of Bridge Edges

Number of bridge edges: 372704

Bridge Edge	Nodes
1	(NodeIndex(1037004), NodeIndex(1037020))
2	(NodeIndex(1710518), NodeIndex(1710537))
3	(NodeIndex(1715523), NodeIndex(1715521))
4	(NodeIndex(717209), NodeIndex(717212))
5	(NodeIndex(766316), NodeIndex(766320))
6	(NodeIndex(1541983), NodeIndex(1542099))
7	(NodeIndex(441990), NodeIndex(441993))
8	(NodeIndex(47059), NodeIndex(47062))
9	(NodeIndex(1682539), NodeIndex(1682540))
10	(NodeIndex(1653505), NodeIndex(1665738))

Bridge edges are edges in the graph whose removal would increase the number of connected components. In the context of the road network, these bridge edges represent critical roads or connections that, if removed, would split the network into separate components, making it impossible to travel between some parts of the network without using other routes.

This finding is significant because it highlights the importance of these 372,704 roads in maintaining the connectivity of the California road network. These critical roads may require special attention regarding maintenance, traffic planning, or disaster recovery plans since their disruption could have a significant impact on the overall network connectivity.

Number of Connected Components

Number of connected components: 2638

A connected component in a graph is a subset of nodes in which there exists a path between every pair of nodes within the subset. In our context, a connected component represents a group of intersections and roads where you can travel between any two intersections within the group without leaving that group.

The fact that there are 2,638 connected components in the California road network implies that the network is divided into 2,638 separate groups, where there is no direct road connection between groups. This could be due to geographical constraints, such as bodies of water, mountains, or protected areas, that prevent roads from being built between certain regions. It could also be a result of limited infrastructure development or zoning regulations.

To improve the connectivity of the California road network, policymakers could consider investing in projects that connect the disconnected components, such as building new bridges, tunnels, or roads. Additionally, transportation planners could study the disconnected components to identify specific regions that may require better public transportation options or improved access to essential services.

Leiden Community Detection

Number of communities: 587120

Community	ID	Size
1	635	4606
2	5825	94
3	16672	87
4	41525	75
5	24395	74
6	8568	71
7	3741	71
8	67105	67
9	25712	62

Community	ID	Size
10	12583	59

The Leiden Community Detection algorithm can help us identify communities in a graph, which are groups of nodes that are more densely connected with each other than with the rest of the network. In the context of the road network, these communities can **represent regions or neighborhoods with strong internal connectivity**.

The output shows that there are a total of 587,120 communities detected in the California road network. This indicates that the road network has many smaller, tightly connected groups of intersections and roads, which could be a reflection of the complex urban planning and varying levels of connectivity across the state.

The table provides a snapshot of the top 10 largest communities in the network, with their respective community IDs and sizes. The largest community (community 1) contains 4,606 nodes, indicating that it is a densely connected area in the road network. The other communities in the top 10 are significantly smaller in size, ranging from 94 to 59 nodes.

These findings can be useful for various purposes, such as:

1. Urban planning and development: Understanding the structure and distribution of communities can help planners identify areas with strong internal connectivity and potentially inform decisions about zoning, infrastructure investments, and traffic management.
2. Emergency response and resource allocation: Knowing the location and size of these communities can help emergency services and local authorities allocate resources more effectively and plan for potential disruptions in the road network.
3. Transportation planning: Identifying well-connected communities can inform the development of public transportation routes and services to ensure efficient connectivity between different regions.

Further analysis could include exploring the geographical distribution of these communities and identifying any potential correlations with demographic or socio-economic factors. This information could help policymakers make more informed decisions regarding transportation planning, infrastructure development, and regional policies.

Core Periphery Node Detection

Category	Quantity	Top Ten Nodes
Number of core nodes	3	NodeIndex(571924), NodeIndex(529105), NodeIndex(504201)
Number of periphery nodes	1965203	NodeIndex(1795590), NodeIndex(1873515), NodeIndex(1165242), NodeIndex(1538243), NodeIndex(1096399), NodeIndex(294281), NodeIndex(1400754), NodeIndex(1619264), NodeIndex(519978), NodeIndex(8853)

In network analysis, core nodes are highly connected nodes that form the backbone of the network, while periphery nodes are less connected nodes that are on the outskirts of the network.

According to the results, there are only 3 core nodes and 1,965,203 periphery nodes in the California road network. This suggests that the network has a few highly connected intersections or hubs, while the vast majority of nodes have relatively fewer connections. This could be attributed to the structure of the road network, which may include a small number of key intersections or highways that connect large regions, while most other roads are local streets with limited connectivity.

The top ten core and periphery nodes are listed in the output. The core nodes (NodeIndex(571924), NodeIndex(529105), NodeIndex(504201)) are likely to represent important intersections, highways, or transportation hubs that play a crucial role in connecting different parts of the network. Understanding the role of these core nodes can be valuable for transportation planning, as well as for preparing for potential disruptions, such as road closures or natural disasters.

The top ten periphery nodes listed in the output are likely to be located in remote or sparsely populated areas with limited connectivity. These nodes may be part of small local road networks that primarily serve the immediate surrounding community. Identifying periphery nodes can help in understanding the distribution of transportation infrastructure and inform decisions about improving connectivity to these areas.

Shortest Path with Landmarks

Given that the shortest path algorithm relies on randomly generated starting and ending nodes, providing a fixed, reproducible output is not feasible. To address this, we have included a sample output in the ‘sample_outputs’ folder. This output demonstrates the results of the shortest path algorithm using a specific pair of randomly generated starting and ending nodes within the California road network.

Users can refer to this sample output to understand the format and interpretation of the results generated by the shortest path algorithm. It is essential to note that the actual results may vary depending on the input data and the random nodes selected. Nonetheless, the sample output provides a general idea of how the algorithm functions and the kind of output it produces.

Please refer to the ‘sample_outputs’ folder to examine the sample output for the shortest path algorithm.

Additionally, the find_shortest_paths function in our implementation includes an “alpha” parameter, which controls the accuracy of the algorithm. You will need to adjust this

value based on your specific needs and requirements. The alpha value influences the trade-off between the algorithm's accuracy and its computational complexity. A lower alpha value may lead to more accurate results but will increase the computation time, while a higher alpha value may speed up the algorithm but could potentially result in less accurate results. The default value for alpha is set to 3.0, but feel free to experiment with different values to achieve the desired balance between accuracy and performance.

AlgoRepo

AlgoRepo is a collection of two graph algorithms that you've written but haven't yet fully integrated into the main program: Label Propagation Algorithm (LPA) and K-Core Decomposition. Below is an explanation of how these algorithms work and the potential implications of their results if they were implemented in the future.

Label Propagation Algorithm (LPA.rs)

The Label Propagation Algorithm is a community detection algorithm that identifies groups of nodes with a high degree of interconnectivity. In the context of the California road network, communities can represent areas with a dense network of roads, such as urban centers or clusters of cities.

The LPA works by assigning an initial unique label to each node in the graph. Then, in each iteration, nodes update their labels based on the most common label among their neighbors. The algorithm continues to iterate until no more updates occur, meaning that the labels have converged.

Upon completion, nodes with the same label are considered part of the same community. The LPA algorithm is fast and scalable, which makes it suitable for large datasets, like the California road network.

The resulting communities could be used to analyze the structure of the road network and identify areas with dense connectivity. Understanding these areas could be helpful for transportation planning, infrastructure investment, and resource allocation.

K-Core Decomposition (kcore.rs)

K-Core Decomposition is an algorithm that identifies the core structure of a graph by recursively removing nodes with a degree lower than a specified value, k. A k-core is a subgraph where all nodes have at least k neighbors within the subgraph. In the context of the California road network, a high k-core could represent areas with highly connected road networks, such as city centers or major highways.

The K-Core Decomposition algorithm first calculates the degree of each node in the graph. Then, it iteratively removes nodes with a degree lower than k and updates the degrees of the remaining nodes' neighbors. The process continues until no more nodes can be removed.

The output of the K-Core Decomposition algorithm is a new graph containing only the nodes that are part of the k-core. Analyzing the k-core can provide insights into the structure and connectivity of the network.

The k-core subgraph could be used to identify areas with strong connectivity in the California road network. This information could be valuable for understanding the structure of the network, identifying critical infrastructure, and informing transportation planning decisions.

Node2Vec Graph Embeddings (graph_embeddings.rs)

Node2Vec is an algorithm that generates vector embeddings for nodes in a graph. These embeddings can be used for various machine learning tasks, such as node classification, link prediction, and community detection. In the context of the California

road network, Node2Vec embeddings could help identify similar regions based on road connectivity or predict future connections between regions.

The Node2Vec algorithm generates random walks on the graph and then uses these walks to learn the embeddings. It employs two hyperparameters, p and q, which control the random walk behavior. The p parameter controls the likelihood of immediately revisiting a node, while q controls the likelihood of exploring nodes farther away from the current node. By adjusting these parameters, the algorithm can generate embeddings that capture various aspects of the graph's structure.

Limitations

1. The California Road Network dataset only represents a portion of the state's road network, and it may not be fully representative of the transportation system as a whole. Future work could involve collecting additional data to better understand the road network and its properties.
2. The road network may change over time, which could affect the accuracy and relevance of the analysis. Future work could involve updating the analysis with more recent data and comparing the results to previous analyses.
3. The analysis focuses on the topological properties of the road network, but it does not take into account other factors that may affect transportation, such as weather, road conditions, or driver behaviour. Future work could involve integrating other types of data into the analysis to provide a more comprehensive understanding of the transportation system.
4. The analysis is limited to the California Road Network dataset, but the methods and techniques developed in this project could be applied to other road networks with similar properties. Future work could involve applying the same analysis to road networks in other states or countries to compare and contrast their properties.
5. The analysis may not capture the full complexity and nuances of the transportation system, and there may be other factors or variables that are important to consider. Future work could involve collaborating with experts in transportation planning and engineering to identify other factors that should be included in the analysis.

Future Work

This project serves as a stepping stone for a more comprehensive endeavor called “AnalyticaHub.” The primary goal of AnalyticaHub is to create a robust platform for analyzing and visualizing large-scale graph datasets effectively. Through this platform, users can gain valuable insights and make data-driven decisions based on the results of their graph analyses.

To achieve this, AnalyticaHub will feature a user-friendly, web-based interface that allows for seamless uploading and analysis of datasets. The platform will also provide a rich set of visualization tools designed to showcase the results in an intuitive and interactive manner.

The technology stack for AnalyticaHub will include:

- **Tauri:** A framework for building lightweight, high-performance applications with web technologies, Tauri will serve as the foundation for the platform, ensuring efficient resource usage and cross-platform compatibility.
- **Svelte:** A modern, reactive JavaScript framework for building user interfaces, Svelte will be used to create the web-based interface, ensuring a smooth and responsive user experience.
- **SurrealDB:** A distributed, real-time database designed for high-availability and scalability, SurrealDB will be utilized for data storage and management, providing reliable and efficient access to large graph datasets.
- **Cytoscape.js:** A powerful graph theory library for visualization and analysis, Cytoscape.js will be employed to render interactive graph visualizations, enabling users to explore and analyze their data effectively.
- **Sigma.js:** A lightweight and flexible library for drawing graphs, Sigma.js will supplement Cytoscape.js in creating visually appealing and customizable graph representations.

Stay tuned for more updates on AnalyticaHub!

Pull Requests & Issues

We always welcome contributions from the community! If you'd like to contribute to the project or have found any issues, feel free to open a pull request or report an issue on our GitHub repository.

Submitting Pull Requests

1. Fork the repository.
2. Create a new branch based on the `main` branch.
3. Implement your changes, additions, or fixes.
4. Make sure to test your code and ensure it follows the project's coding style and guidelines.
5. Commit your changes, and write a clear and concise commit message describing what you've done.
6. Push your branch to your forked repository.
7. Create a new pull request, and be sure to provide a thorough description of your changes.

Reporting Issues

If you encounter any problems or have suggestions for improvements, please don't hesitate to open a new issue on our GitHub repository. When reporting an issue, try to provide as much relevant information as possible, including:

- A clear and descriptive title.
- A detailed description of the issue or suggestion.
- Steps to reproduce the problem (if applicable).
- Any relevant error messages or logs.
- Your operating system and software versions.

Your feedback helps us make the project even better, and we truly appreciate your support! 🙌

License

Acknowledgements

We would like to express our gratitude to:

- Professor Leonidas Kontothanassis from the BU Faculty of Computing & Data Sciences, for his invaluable guidance and support throughout the development of this project.
- The Stanford Network Analysis Project (SNAP) for providing the dataset used in this project. Their extensive collection of network datasets and resources has been instrumental in the development and validation of our work.