CMSE 831 Semester Project

Maximum Flow Optimization in the California Road Network

Angelica Gacis, Yena Hong, Anna Jeffries, Chaeyeon Yim

December 10, 2023

Contents

1 Overview

We have chosen to pursue a project concerning maximum flow optimization. While maximum flow is a classic optimization topic, a brief survey of extant literature shows that much of the work is abstracted and theoretical in nature. To draw a contrast to this, we became curious and wanted to see what would happen if we applied maximum flow optimization in the context of emergency routes and community evacuation. We will also compare different methods to see which produces the most viable results.

1.1 Maximum Flow Optimization

The maximum flow problem is a fundamental challenge in network theory, where the goal is to find the highest possible flow from a starting point (source) to an end point (sink) in a network. The flow must not exceed the limits (capacities) of the network's paths. This problem is important both in abstract mathematical research and real-world applications, such as traffic management and network design. It's also a key area in computer science, where finding better algorithms to solve this problem can lead to improvements in many different systems that depend on optimizing how resources are distributed. The scope of the overall question is broad, but we will try to investigate the small point we have chosen.

1.2 Background

As global warming worsens, it becomes increasingly important to address its impact on people in vulnerable areas. In particular, disaster preparedness and crisis management are crucial to address natural disasters. In California, which is proximal to the Ring of Fire, climate change poses a significant threat to the safety of millions of people who live along the state's coast.[1] Increasing volcanic activity and rising sea levels, driven by global warming, pose a regular risk to the coastal areas of California, making them more susceptible to frequent and severe natural disasters such as floods and tsunamis.[2] Moreover, wildfires in California are growing in size and intensity, resulting in the loss of extensive inland areas. These facts underscore the urgent need to confront the consequences of climate change in the region.

Therefore, considering maximum flow optimization for California's roads can be beneficial in many aspects. With an efficient flow strategy, the state government and policymakers can address the challenges posed by climate change and its impact on residents' safety,

[1] NOAA Office for Coastal Management.

[2] Id.

especially in the context of natural disasters and evacuations. More specifically, it can contribute significantly to improving the state's resilience to the effects of climate change, enhancing emergency response systems, and safeguarding the well-being of its citizens in the face of evolving environmental challenges.

1.3 Problem Statement

As stated above, with the threat of natural disasters consistently increasing each year due to climate change, residents in California face heightened risks. A proactive and effective strategy is needed to take action in advance and safeguard public safety. In this vein, we aim to develop a maximum flow optimization method for the California road network database to aid in developing an ideal evacuation plan by discovering optimal paths.

2 Data

Two datasets, *California Road Network's Edges* and *California Road Network's Nodes*, were chosen in order to implement the max flow optimization method in geographic networks; the datasets contain network and geographic information for the state of California.

2.1 California Network Data

The dataset *California Road Network's Edges* provides a detailed view of the connections between various nodes within the road network. Each of the 21,692 edges is identified by a unique Edge ID and includes both a Start Node ID and an End Node ID, delineating how different points in the network are interconnected and inducing a directed graph. This information is crucial for understanding the topology of the network. Additionally, each edge includes a vital measurement termed "L2 Distance," which represents the Euclidean distance between the start and end points of each edge. This distance information is important for calculating the shortest paths, estimating travel times, and analyzing the spatial distribution of the road network.

2.2 Geographic Information

The dataset *California Road Network's Nodes* comprises a series of entries, each designated by the unique node IDs that appear in the *Edges* dataset. For every node, the corresponding geographic coordinates—expressed in decimal degrees of longitude and latitude—are included. These coordinates pinpoint the precise east-west (longitude) and north-south (latitude) positions of each node, essential for geographic mapping and spatial analysis.

This information allows us to "map" the network with precise geography (as opposed to the typical arbitrary representations of a network without placement).

2.3 Preprocessing

2.3.1 Integrating Geographic Location Information

Taking the two datasets mentioned, we merged them together by using the node ID as the key. Because each entry in the *Edges* dataset includes both the Start Node and End Node for each edge, we appended the longitude and latitude for both nodes.

2.3.2 Calculating Capacity

One key omission of our selected data is the lack of capacity values for each edge. Therefore, we have formulated an estimated road capacity calculation for each edge, following a set of defined assumptions. Initially, we assume the average vehicle length to be 5 meters, typical of a standard passenger car. Furthermore, we adopt a safe following distance of 2 seconds, in line with conventional safety norms. The average speed is taken as 13.9 meters per second, corresponding to 50 km/h, which is a common speed on urban roads. Using these values, our formula to calculate the road capacity is given by:

$$\text{Road Capacity} = \frac{\text{L2 Distance}}{\text{Average Vehicle Length} + (\text{Average Speed} \times \text{Safe Following Distance})}$$

The goal for our capacity calculation lies in determining the total space a single vehicle occupies, which encompasses the length of the vehicle and a safe following distance (the latter converted into meters based on the average speed). We then estimate the road capacity by dividing the L2 distance of each road segment by this total space requirement per vehicle. This methodology provides a practical way to assess the maximum number of vehicles that can be accommodated on each road segment.

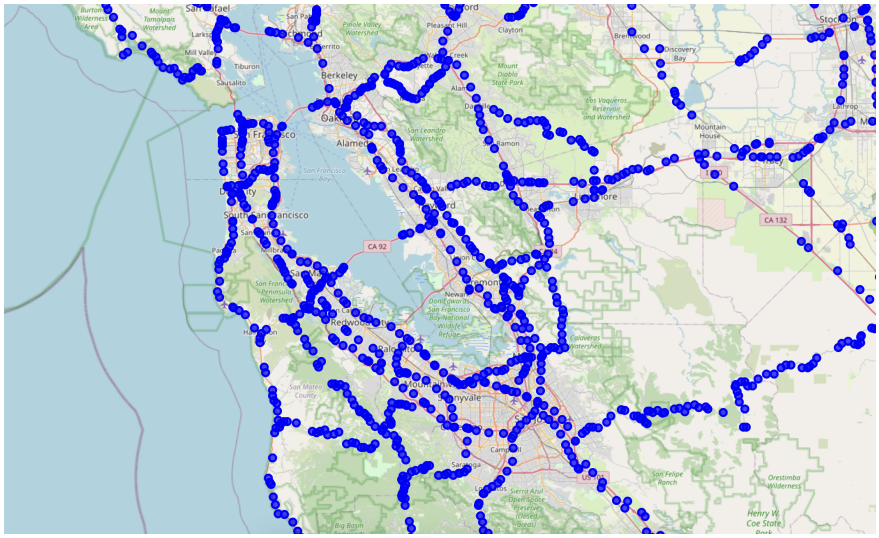

Figure 1
California network overlaid on a map using geographic information

2.4 Sub-setting the Data

Due to the large volume of data, we initially selected a subset to analyze with the intention to later scale up our process. Our initial criteria for the subset of data depended on whether a given continuous directed path shared starting or ending nodes with another continuous directed path. For this exercise. we looked only among the first 100 entries. This concept is illustrated in the following example, where the paths between nodes 1, 2, and 3 are chosen but the paths with 4 and 5 are excluded.

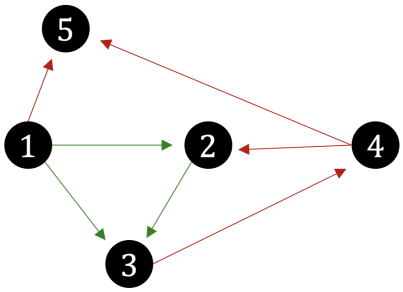

Figure 2
Example of initial subsetting

The image below represents this subset of nodes as a graphical network. At first glance, it appeared as if all paths were interconnected.

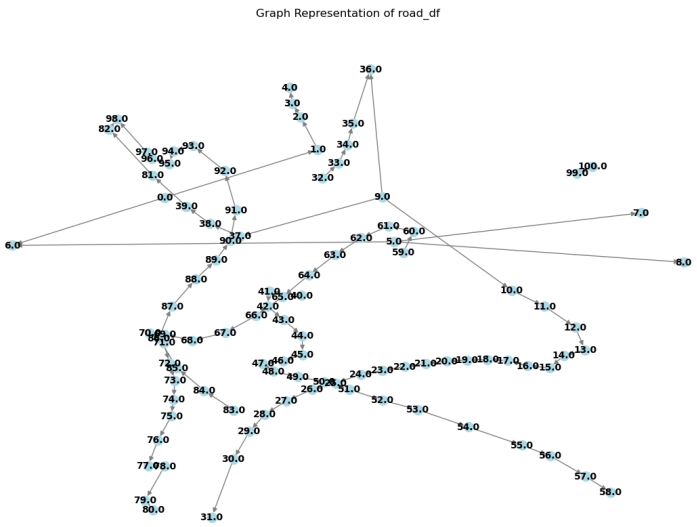

Figure 3
Fully connected paths (without geography)

However, upon re-visualizing the network with geographic information, we observed that the graph was actually composed of five distinct groups.

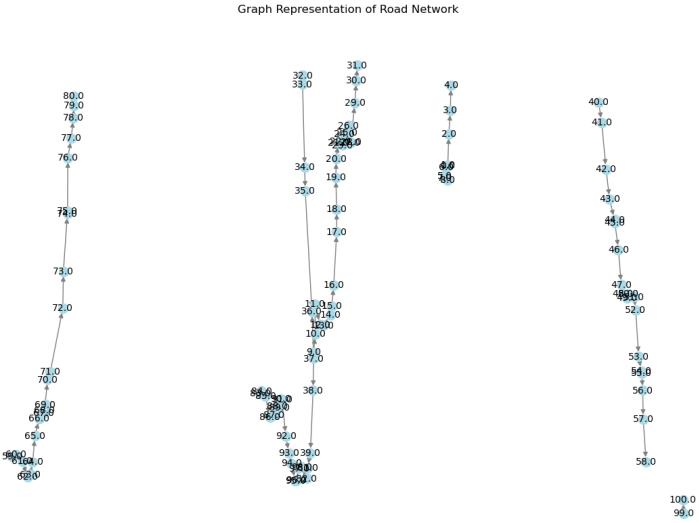

Figure 4
Fully connected paths (with geography)

We still chose to conduct maximum flow optimization on this group, despite its disjointness, because we wanted to compare its results to those of other selected subsets. As indicated by the red arrows in the figure below, we were able to identify a path of maximum flow from node 71 to 74. This exercise is obviously quite limited, though, due to the lack of well-connected paths across the entire subset network.

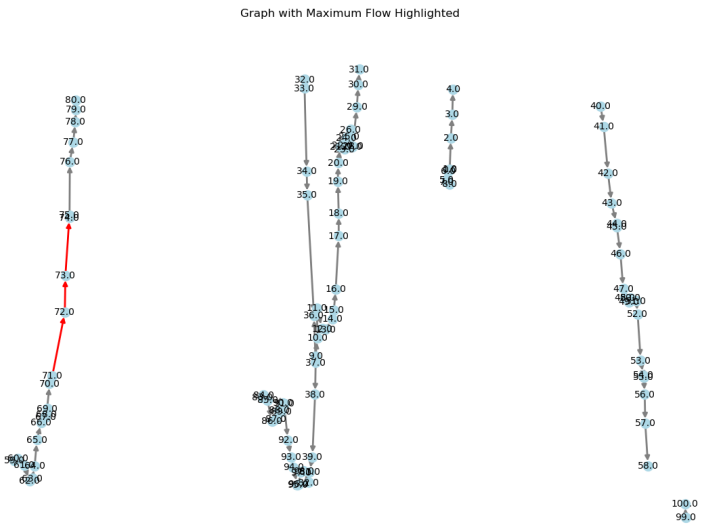

Figure 5
Max flow optimization on fully connected paths (with geography)

We also tried pruning the data in different ways in an attempt to apply representation learning. One way was to limit the nodes to those that are only connected to paths with at least 5 other nodes present. Another way was to limit the nodes which were connected to edges with capacities that were in the top 10th percentile of the capacity range. However, these efforts resulted only in several iterations of what we will call "weakly-connected paths." That is, where paths were disjoint (as illustrated in the above example) and/or where paths were interrupted by the limitations of the initial imposed directionality of the graph. We will discuss these below.

2.5 Missing Paths

As observed in the previous example, we recognized that the network data contains unconnected paths. To address this issue, we employed two strategies. The first is to impose bidirectionality on all paths, and the second is to exclusively search for "well-connected paths."

2.5.1 Bidirectionality On All Paths

In a real disaster situation, people are likely to ignore the normal direction of road travel in their efforts to escape from the disaster area. Our strategy to impose bidirectionality on all paths not only reflects this realistic scenario but also, by implementing bidirectionality, allows us to maximize the use of our data to create a more extensively connected network.

In order to achieve global bidirectionality in our network, we simply took the inverse of

every entry according to its start and end node ID—effectively flipping the associated path to create a symmetric edge with the opposite direction—and appending it to the original dataframe. While we considered transforming our network to be entirely undirected—that is, with no directions associated with any edges—we opted to pursue this approach instead, as many calculations and algorithms associated with maximum flow optimization require directed edges.

2.5.2 Exclusively Well-connected Paths

This strategy involves selecting paths from our data that have the highest number of interconnected nodes, building a network based on these connected paths, and then predicting the maximum flow within this network. Effectively, we sought out the most dense portions of the original network that could be connected together base on their original edge directions.

Our algorithm for this strategy follows:

1. **Initialization**: The process begins with the creation of an empty dataframe, `largest_subset`, intended to store the largest group of connected nodes found. Additionally, a counter `largest_subset_size` is set to zero, serving as a benchmark for the largest subset size identified during the analysis.
2. **Data Iteration**: The analysis involves iterating over each unique start node in the dataset. For each node, a subset (`current_subset`) is created, encompassing all connections originating from that specific node.
3. **Subset Expansion**: Two sets, `nodes_to_check` and `checked_nodes`, are maintained to track the nodes that require exploration and those that have been already explored, respectively. This ensures an efficient and non-repetitive exploration process. The subset is expanded by adding new connections and updating the nodes to be checked.
4. **Identification of Largest Subset**: Post exploration of each start node, the size of the current subset is compared with the largest subset size recorded. If larger, `current_subset` is designated as the new `largest_subset`, and its size is updated in `largest_subset_size`.
5. **Finalization and Output**: Upon completion of the iterations, the largest subset found is stored in `largest_subset`. The index of this DataFrame is reset for clarity. The final output includes a display of the initial entries of the largest subset and the total size of this subset.

As a result of our efforts, we determined that the best well-connected path is a network consisting of a total of 374 nodes. The figures below illustrate this network that we identified.

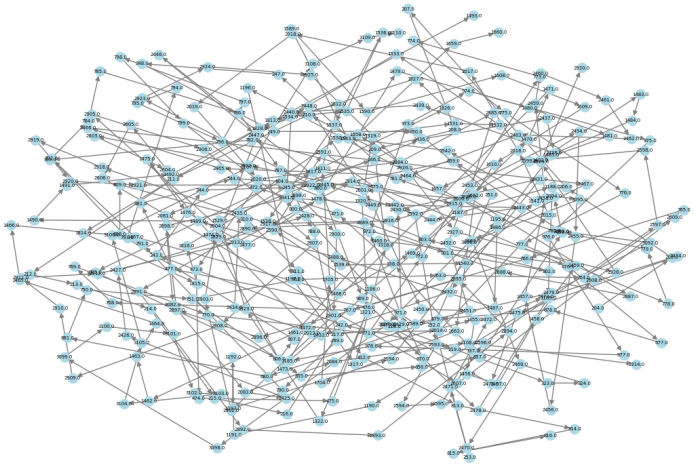

Figure 6
Well-connected paths (without geography)

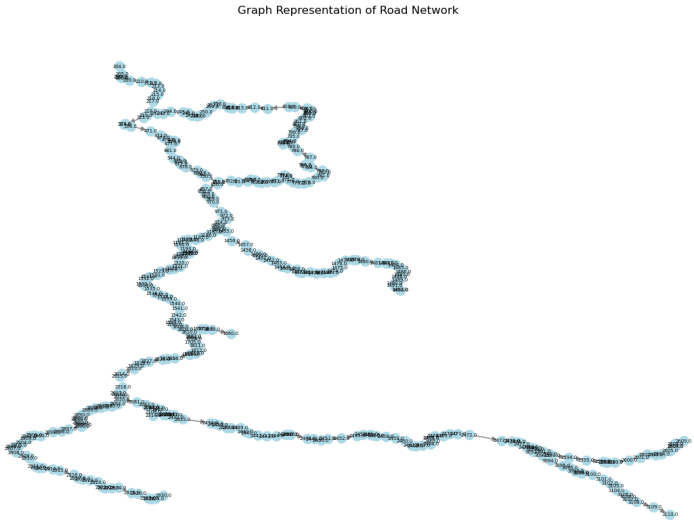

Figure 7
Well-connected paths (with geography)

2.6 Summary

In summary, we find ourselves with two usable datasets out of our three experiments, as well as the original data itself.

- The data as-is, with the addition of the capacity calculations. Let us denote this as dataaset A.
- The data with imposed bidirectionality, with the addition of capacity. Let us denote this as dataset B.

- The subset of exclusively well-connected paths, with the addition of capacity. Let us denote this as dataset C.

As mentioned previously, our initial approach to sub-setting the data proved unfeasible because of the disjoint paths it created. Therefore, we will focus our attentions on the three datasets that we believe may yield valuable results.

3 Simulations

3.1 Optimization Problem

Let $N = (V, E)$ be a network with $s, t \in V$ being the source and the sink of N, respectively. The function $f : E \to \mathbb{R}$ is defined as $f_{u,v}$ $\forall (u, v) \in E$. Then the optimization of this function looks like:

$$\max(|f|) = \sum_{v:\ (s,v)\in E} f_{s,v} = \sum_{u:\ (u,t)\in E} f_{u,t}$$

This is the classic statement of a maximal flow optimization problem where we are maximizing f with flow going in and out of every node. However, for our purposes, we are interesting in maximizing f *while also* minimizing the L2 distance given in our dataset.

While we do not have any insight has to how exactly the given L2 distance was originally calculated, we will assume that, using the nodes in V, it would be $d_{s,t} = \sqrt{(x_s - x_t)^2 + (y_s - y_t)^2}$ for s and t represented by coordinates (x_s, y_s) and (x_t, y_t). More generally, the function d can be expressed as:

$$D = \sum_{s\in V} \sum_{t\in V} d_{s,t} \quad \text{and} \quad \min(D)\ \forall s, t \in V$$

A further constraint is that every road (edge) in the network has a capacity (calculated in §2.3.2), which can be interpreted as the maximum number of cars that can travel on that road without causing a traffic jam. The flow of cars $f_{u,v}$ on any road from u to v must not exceed this capacity $c_{u,v}$. Formally, this is represented as $f_{u,v} \leq c_{u,v}$ $\forall u, v \in E$.

3.2 Methods

3.2.1 Preflow-Push (Push-Relabel) Algorithm

The Preflow-Push (also known as the Push-Relabel) algorithm is a method used in solving the maximum flow problem and is especially effective for sparse networks. Unlike algorithms

that focus on finding augmenting paths directly (discussed below), the Preflow-Push algorithm operates by initially pushing as much flow as possible through the network. It then iteratively adjusts and redistributes this flow, a process known as "relabeling," to discover the optimal flow distribution. This approach is more localized, emphasizing individual node potentials and the redistribution of flow across the network. Its efficiency shines in scenarios where the network has fewer edges relative to the number of nodes.

The standard time complexity is $\mathcal{O}(|V|^2|E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the network. However, it can be improved to $\mathcal{O}(|V|^2\sqrt{|E|})$ when the algorithm consistently chooses the highest-labeled node each time to push the flow, adopting a triage approach. This makes it faster for sparse networks with fewer edges as the square root of the edge count can significantly reduce the number of operations needed. The latter of these approaches is what is used by `NetworkX`.[3]

3.2.2 Edmonds-Karp Algorithm

On the other hand, the Edmonds-Karp Algorithm follows a more traditional approach in the wake of the Ford-Fulkerson Algorithm. It is based on using a Breadth-First Search (BFS) to find augmenting paths in the network. This method systematically explores the network level by level and chooses the augmenting path with the least number of edges. The Edmonds-Karp Algorithm is particularly advantageous in dense networks, where the number of edges is large in comparison to the number of nodes, as this provides the algorithm with an abundance of multiple path options. The time complexity is $\mathcal{O}(|V||E|^2)$.[4] This reflects the algorithm's exhaustive search for augmenting paths and is particularly noteworthy in dense networks where the edge count is high.

3.2.3 Shortest Augmenting Path (Dijkstra's) Algorithm

This algorithm, while similar to the Edmonds-Karp, differs because it specifically seeks out the *shortest* possible augmenting path, which aligns with our desire to consider the distance of each edge whilst maximizing flow. This algorithm's time complexity is $\mathcal{O}(|V|^2|E|)$.[5]

---

[3] See `NetworkX` documentation.

[4] See `NetworkX` documentation.

[5] See `NetworkX` documentation.

3.3 Limitations

Applying the maximum flow algorithms to real-world networks like the California Road Network graph involves navigating significant challenges. In practical networks, not all paths are necessarily connected, which poses a problem for algorithms that assume full connectivity. This issue is similar to having missing values (`NA`s) in a data frame, where no direct path exists between two points. While strategies such as imposing bidirectionality on all paths or focusing on well-connected components can mitigate these issues, the complexities of real-world network structures can still lead to sub-optimal solutions.

For instance, in complex networks, fairness, efficiency, and the network's structure play crucial roles in the throughput and allocation of flow, affecting the location of bottlenecks. The real-life applicability of congestion control algorithms has broad implications, especially as society becomes increasingly reliant on stable and efficient infrastructure networks. The need for robust congestion control is heightened by the potential catastrophic consequences of network failure, as illustrated by historical network collapses. Additionally, the lack of population information associated with our data is a particular hinderance, as such information would help us assess the volume that must move through the network.

Therefore, while maximum flow algorithms provide a theoretical foundation for understanding flow through a network, the intricate and varied nature of real-world networks like California Road Network requires nuanced approaches that account for the unique characteristics and potential disconnects within the network's structure. Nevertheless, we will endeavour to explore as much application as possible in this context.

4 Results

4.1 The Data As-is

Here, we apply all three previously mentioned algorithms to the data as-is, with the addition of the capacity calculations.

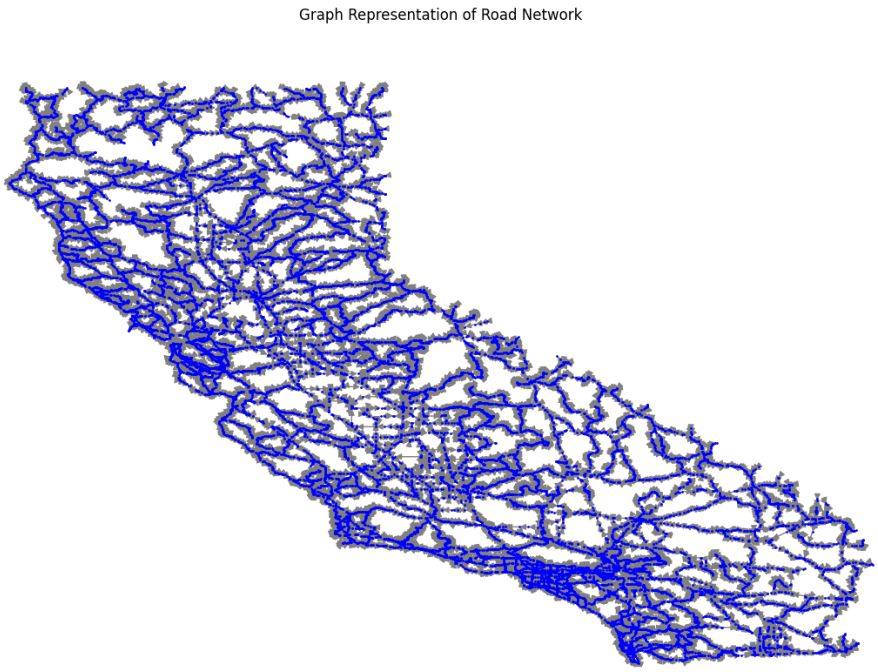

Figure 8
The entirety of paths with geography

In order to define our sources and sinks, using latitude and longitude, we identified several key urban areas of interest, including Oakland and Eureka. We designated the nodes within these areas as the sources and allowed anything outside of the boundary to be considered a sink.

However, due to the computational load, we found that we also had to limit the area of our sinks. Instead of allowing sinks to be anything outside our selected areas, we created another window in which to limit possible sinks, around the window of area we have already selected as our sources.

4.1.1 Eureka

```
# Python code
min_latitude, max_latitude = 40.1, 41
min_longitude, max_longitude = -124.6, -124
# Same code as before to make initial df_oakland
# Same code for excising df_eureka contents from whole df
df_sans_eureka_small =
    df_sans_eureka[(df_sans_eureka['Start_Latitude'].between(min_latitude-.5,
    max_latitude+.5)) &
```

```
    (df_sans_eureka['End_Latitude'].between(min_latitude-.5,
        max_latitude+.5)) &
    (df_sans_eureka['Start_Longitude'].between(min_longitude-.5,
        max_longitude+.5)) &
    (df_sans_eureka['End_Longitude'].between(min_longitude-.5,
        max_longitude+.5))
] # a bit bigger of a window since it is overall a less dense area
```

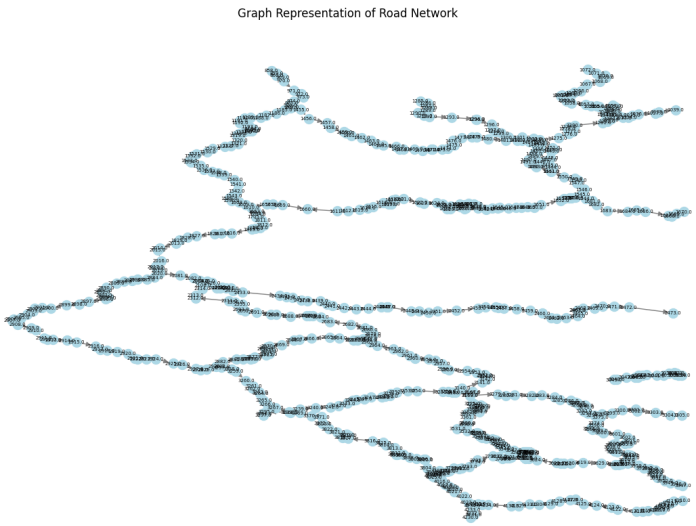

Figure 9
The window of area around Eureka, CA

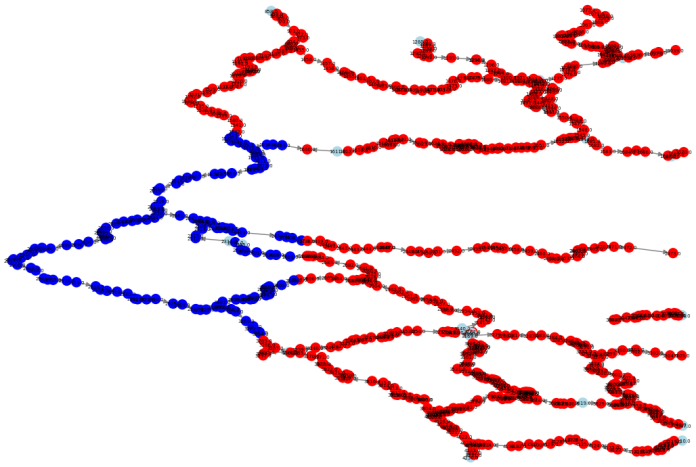

Figure 10
The window of area with the sources highlighted in blue and sinks highlighted in red

Edmonds-Karp

While the Edmonds-Karp algorithm ran successfully with a runtime of about 52 minutes, it appears no path was found. While disappointing, this is not entirely unexpected.

Preflow-Push

Though with the fastest runtime of any of the three algorithms, no path was found. This may also be deduced by the lack of paths found previously with the Edmonds-Karp attempt.

Shortest Augmenting Path

We chose not to run this particular algorithm because it would yield the same results as our previous two attempt; the presence (or lack thereof) of a viable path given the constraints of a directed graph is an immutable feature of this particular dataset used.

4.1.2 Oakland

```
# Python code
min_latitude, max_latitude = 37.7, 37.9
min_longitude, max_longitude = -122.4, -122.1
# Same code as before to make initial df_oakland
# Same code for excising df_oakland contents from whole df
# Make a subset for sinks window
df_sans_oakland_small =
    df_sans_oakland[(df_sans_oakland['Start_Latitude'].between(min_latitude-.5,
    max_latitude+.5)) &
            (df_sans_oakland['End_Latitude'].between(min_latitude-.5,
                max_latitude+.5)) &
            (df_sans_oakland['Start_Longitude'].between(min_longitude-.5,
                max_longitude+.5)) &
            (df_sans_oakland['End_Longitude'].between(min_longitude-.5,
                max_longitude+.5))
        ]
```

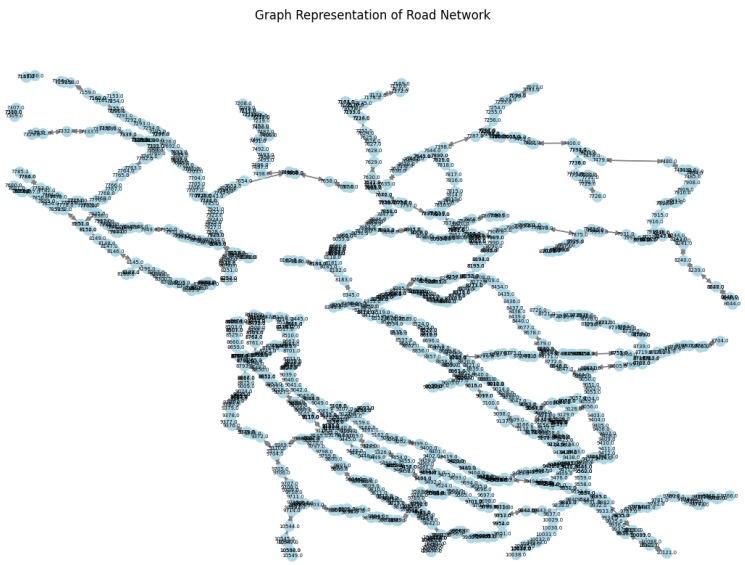

Figure 11
The window of area around Oakland, CA

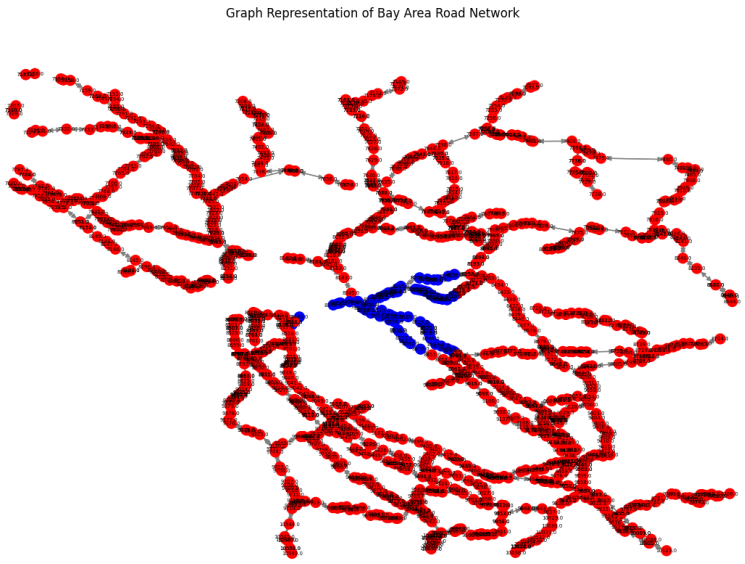

Figure 12
The window of area with sources highlighted in blue and sinks highlighted in red

No paths were found using any of the algorithms.

It should be noted that the odd layout of this particular graph is due to the fact that the capturing window includes San Francisco in the bottom left area. This complicates matters, but for the sake of brevity we are not going to address this peculiarity. We will also note that San Francisco itself sits on elevation, whereas Oakland does not, so we consider our

approach here to not be unreasonable.

4.2 The Data with Bidirectionality

Here, we apply all three previously mentioned algorithms to the data with bidirectionaliy imposed and the addition of the capacity calculations (§2.5.1). Here we note that some aspects were complicated in this approach due to source and sink nodes overlapping around the edges of intersection between the designated source and sink areas. For the sake of brevity, we adjusted the algorithm to pass on any instances where it encountered `source == sink`. Otherwise, the area selection remains unchanged from that in the previous section.

4.2.1 Eureka

Edmonds-Karp

The algorithm ran and produced results with a run time of approximately 24 minutes (presumable due to the area's relatively low density).

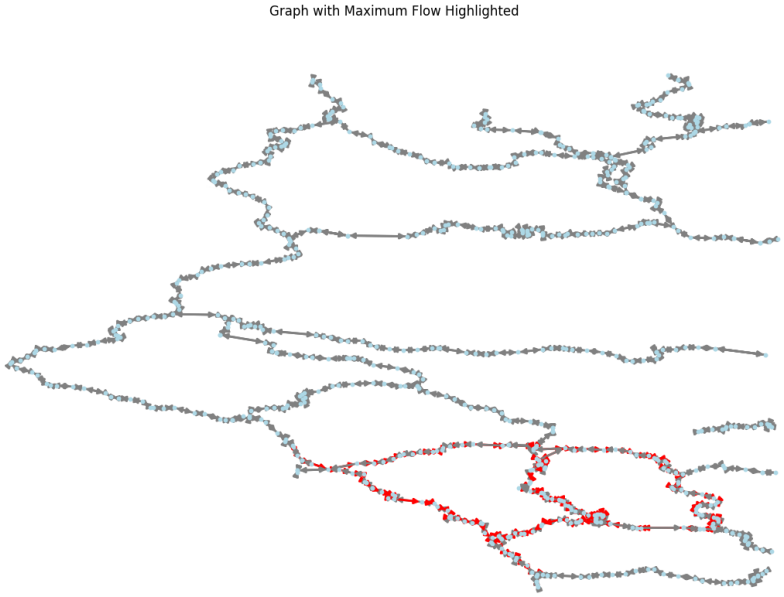

Figure 13
The resulting graph with maximum flow highlighted in red

The maximum flow value the algorithm achieved was **14.1067**.

Preflow-Push

The algorithm ran and produced results with a run time of approximately 170 minutes.

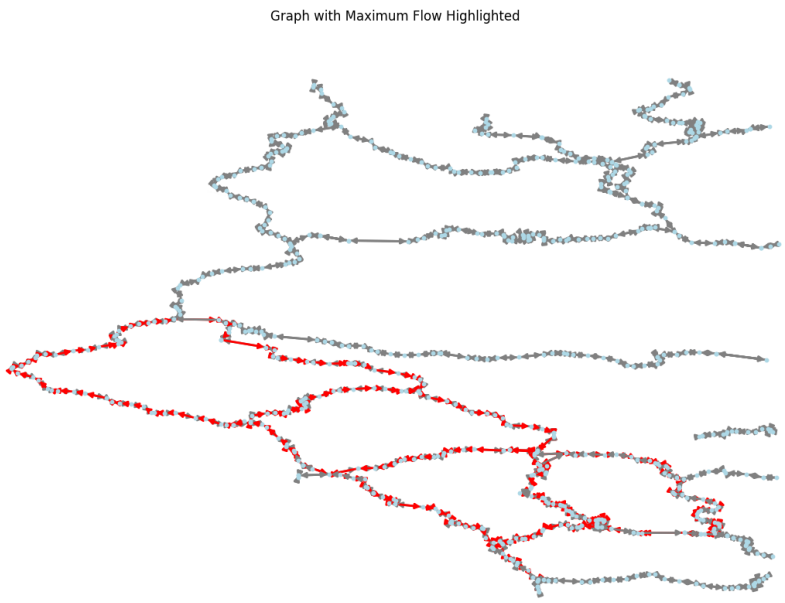

Figure 14
The resulting graph with maximum flow highlighted in red

The maximum flow value the algorithm achieved was **14.1067**. While the maximum flow value is the same as with the Edmonds-Karp method, one will observe that the path highlighted is significantly broader than the previous path. This larger path includes the path found by the Edmonds-Karp method. Out of the three iterations using this particular subset of data, we believe the Preflow-Push method produced the most meaningful results. While costly in terms of computation time, it gives us the most "realistic" results, as it were, because of the broader paths it incorporates, including a significant number of those that are in our "critical region" from which we are simulating an evacuation.

Shortest Augmenting Path

The algorithm ran and produced results with a run time of approximately 201 minutes.

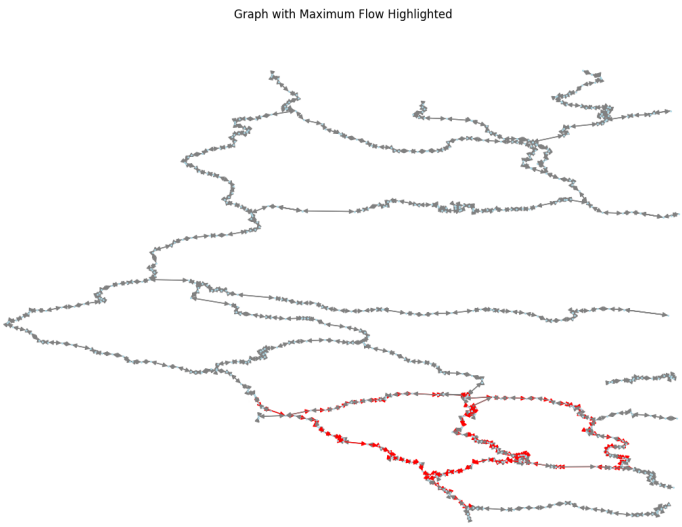

Figure 15
The resulting graph with maximum flow highlighted in red

The maximum flow value the algorithm achieved was **14.1067**. Here we see that the results and path are the same as those using the Edmonds-Karp method, despite a significantly longer run time.

4.2.2 Oakland

Edmonds-Karp

The algorithm ran and produced results with a run time of approximately 64 minutes

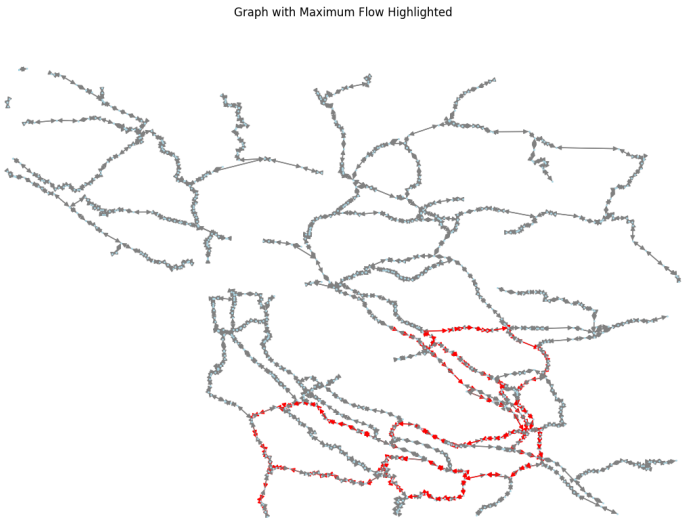

Figure 16
The resulting graph with maximum flow highlighted in red

The maximum flow value the algorithm achieved was **8.8231**.

Preflow-Push

The algorithm ran and produced results with a run time of approximately 353 minutes

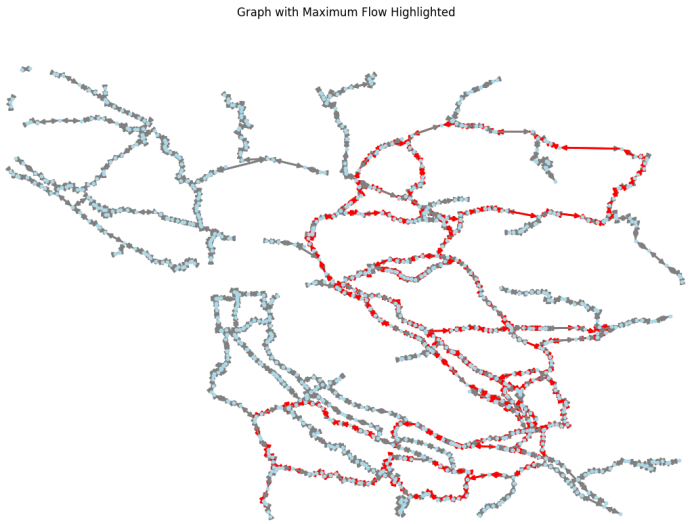

Figure 17
The resulting graph with maximum flow highlighted in red

The maximum flow value the algorithm achieved was **8.8231**. Much like we saw previously with Eureka, the maximum flow value is the same as with the Edmonds-Karp method, but the path high- lighted is significantly broader than the previous path. This larger path includes the path found by the Edmonds-Karp method. While costly in terms of computation time, it gives us the most "realistic" results, as it were, because of the broader paths it incorporates, including a significant number of those that are in our "critical region" from which we are simulating an evacuation.

Shortest Augmenting Path

The algorithm ran and produced results with a run time of approximately 380 minutes

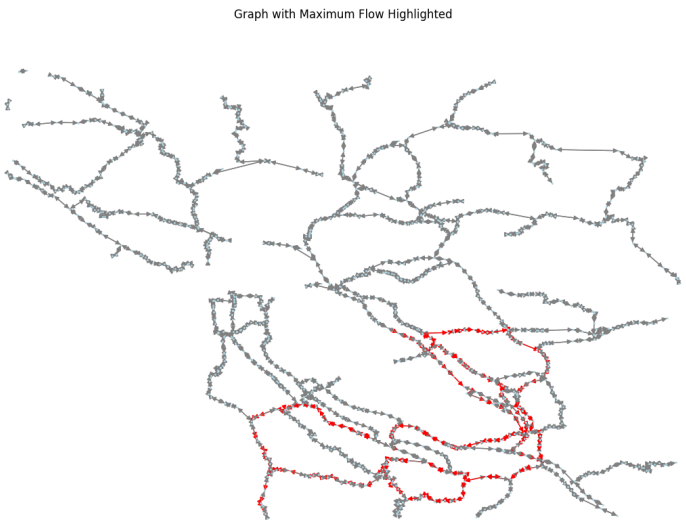

Figure 18
The resulting graph with maximum flow highlighted in red

The maximum flow value the algorithm achieved was **8.8231**. Such a significant run time for the exact same results in value and path discovery as the Edmonds-Karp method—which only required a sixth of the same time—makes this the worst choice of the three methods for this particular subset.

4.3 Well-connected Paths

We analyzed the maximum flow from the coastal area (blue zone) to the inland region (red zone) with the largest subset of well-connected paths we identified earlier in §2.5.2, producing one usable dataset still respecting the original directions on the graph (sans imposed bidirectionality).

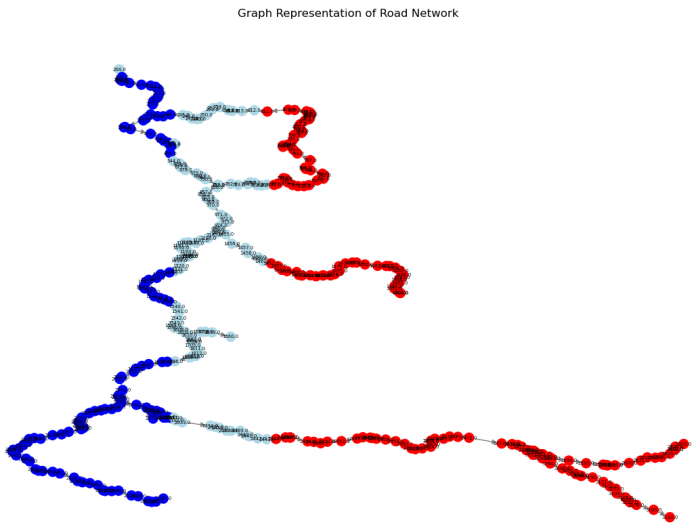

Figure 19
Graph of the largest continuous well-connected path in the entire dataset

As a result of this analysis, we were able to find an optimized maximum flow path as shown in the figure below.

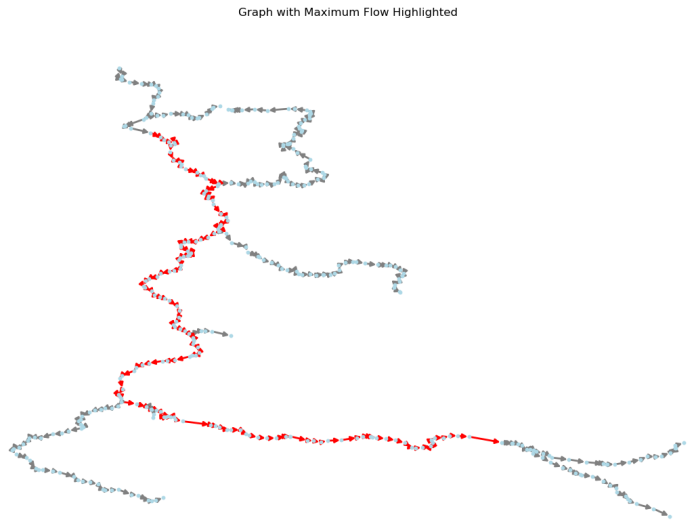

Figure 20
The resulting graph with maximum flow highlighted in red

The maximum flow achieved was **2.5274**. While we are not entirely certain about the exact geographic details of the path, based on a cursory review of the longitude and latitude, it appears to largely be in the northwest corner of the state which is mostly made up of National Forest land. In that event, this result is not particularly useful for our purposes as it is not relevant to our goal of assessing coastal areas.

5 Summary

From what we were able to achieve, we can see that, in our use case, the Preflow-Push algorithm yielded the best results, though with the burden of significant run times. It is unclear if this difference in performance would remain true when applied on a larger scale. On one hand, the Calfornia Network Data is quite dense, but the multiplicity of edges and nodes do not define very many unique paths, in the sense that they do not traverse a large percentage of the land's area.

We would have liked to apply the maximum flow optimization algorithms to more geographic windows, but found ourselves hampered by computational limits. We had previously included a selection of the area of Long Beach, but it quickly became evident that, while the window did not cover much area, it was also very dense very dense, hence the incredible amount of time that would be required to run the algorithms.

We want to acknowledge and recognize the flaws and shortcomings that we encountered during the month of time we spent on our project. Given more time and computing resources, we found this endeavour quite interesting and would have pursued it further if afforded the opportunity.

In conclusion, while largely a learning experience without especially concrete or satisfactory results, we are reasonably pleased with the outcome. Being wholly unfamiliar with network optimization and its nuances prior to this project, we effectively had to learn everything from scratch in a month's time, so we are gratified by what we were able to accomplish.

6 References

Buzna, Ľ., & Carvalho, R. (2017). Controlling congestion on complex networks: fairness, efficiency and network structure. Scientific Reports, 7(1), 9152. https://doi.org/10.1038/s41598-017-09524-3

Edmonds, J., & Karp, R. M. (1970). Theoretical improvements in algorithmic efficiency for network flow problems. Operations Research Center Report ORC 70-24, University of California, Berkeley, 1-9.

Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. Journal of the ACM, 35(4), 921–940. https://doi.org/10.1145/48014.61051

Teng, S.-H. (2023). Technical perspective: Maximum flow through a network: A storied problem and a groundbreaking solution. Communications of the ACM, 66(12), 84. https://doi.org/10.1145/3623277