Research Report

Introduction

This summer I worked on the Delineo Project,a virus simulator. To simulate a population of people we should know who they come into contact with. For example, a simulation of a town should also include places the people of the town visit, and include places where people visit the town from. In an ideal simulation all people will be accounted for.

I was responsible for the clustering algorithm, which takes a starting geographical area (can have shops, restaurants) called a Census Block Group (CBG), and repeatedly adds more CBGs until a population limit is reached. The goal is to have as much movement between the chosen CBGs and as little movement between the chosen CBG's and outside, such that the chosen CBG's are as isolated as possible. To do this, SafeGraph movement data was used, and US Census Bureau Shapefiles were used for visualization.

Graph Construction

Each CBG has Points of Interest (POI's) which can be restaruants, shops ect. Safegraph has data on these POI's such as which CBG it belongs to, the median time spent in the POI and from which CBG the POI was visited, and how many people from that CBG visited the POI.

To construct the graph, we let each CBG be a node. To find the weight of an edge between a CBG1 and CBG2, we find which POI's in CBG1 CBG2 visited, and for each POI we multiply the median dwell time by the amount of visitors from CBG2, and then add these across the POI's in CBG1. We then repeat for visits from CBG1 to CBG2, add these 2 directed weights and create 1 undirected weight between them.

We multiply by the median dwell time since spending more time in an area will lead to higher chance of infection and should be accounted for with greater weight.

Map Visualization

We obtained CBG shapefiles from the US Census Bureau and mapped the CBGs using Folium. CBGs are color coded according to this ratio: $(movement\ with\ S)/(movement\ with\ S + movement\ out\ of\ S)$. Where movement is the movement of the CBG, and S is our cluster of CBGs.

Color Code:

- Blue: Ratio > 0.8
- Green: Ratio > 0.6
- Yellow: Ratio > 0.4
- Orange: Ratio > 0.2
- Red: Ratio > 0

We also created an animated map to show the progression of S, and how the ratio of CBGs in S change as more CBGs are added.
Animated map of Hagerstown:
https://www.youtube.com/watch?v=GQlaK0gMoi4&ab_channel=Ryad

Algorithms

A good set of CBGs (called set S) is isolated, to measure this we used this ratio:

$$(movement\ in\ S)/(movement\ in\ S + movement\ out\ S)$$

Where movement is the movements of the CBGs in S.

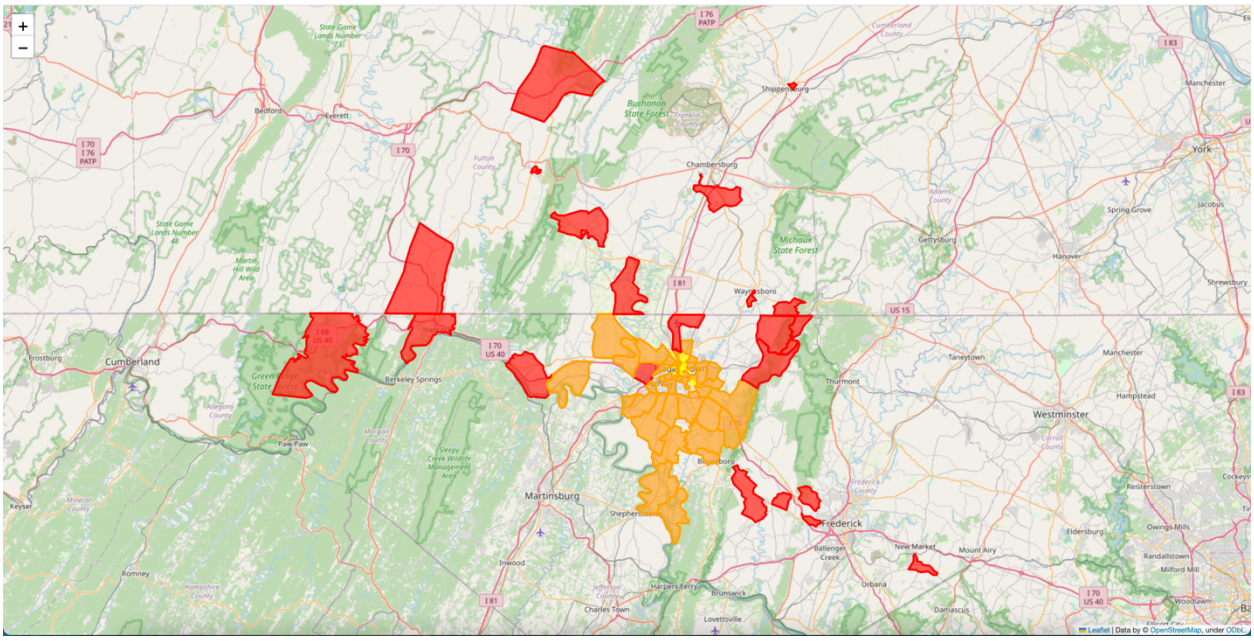
I tried 3 algorithms with the purpose of maximizing the ratio above. The first was Greedy Ratio.

Greedy Ratio:

Since we wanted the highest ratio, I tried the greedy approach of maximizing the ratio at every step. (Every time we add a new CBG).

At each iteration, the algorithm would look at all adjacent CBG's to the entire set S, and calculate what the ratio would be if we added each one individually. It would then add the CBG that caused the highest ratio and then continue until a population limit was reached.

While this approach sounds okay, it was shortsighted and formed bad clusters.



(Greedy Ratio on Hagerstown, min population 5000, Ratio = 0.26)

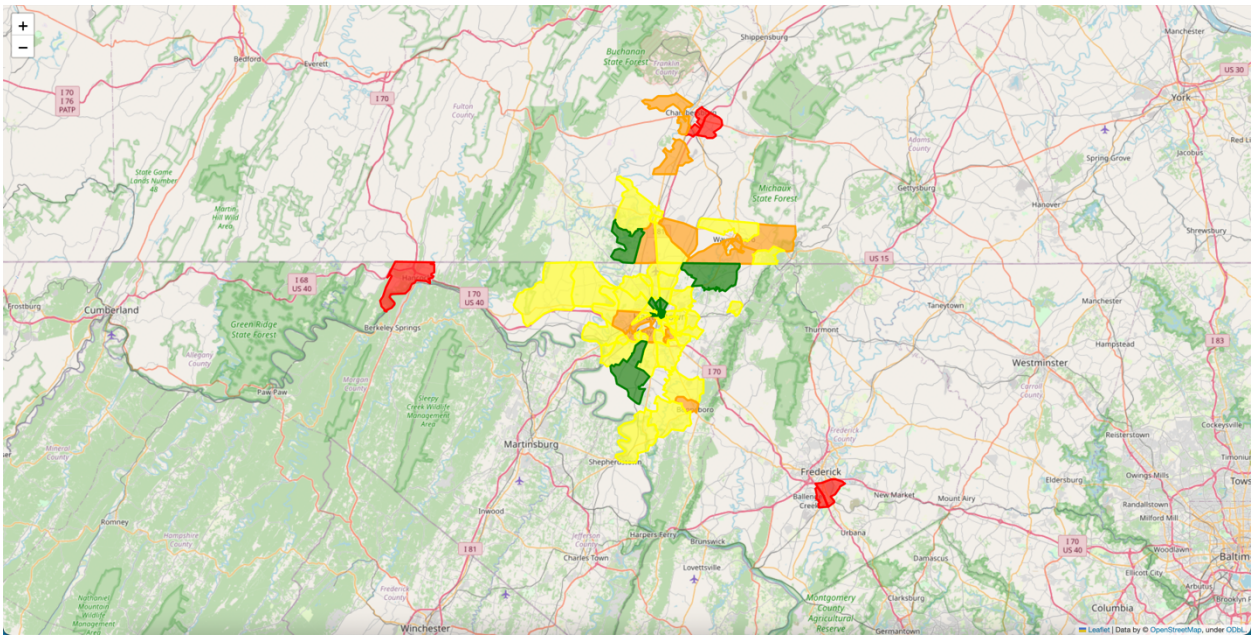
<u>Greedy weight:</u>

This algorithm finds the CBG with the most movement with set S and adds the CBG to S. This continues until we reach the minimum population.

This approach has good results, however if we are clustering on a rural CBG that is next to a city, the algorithm gets 'sucked' into the city, and clusters on the city instead of our seed CBG. To combat this, we tried modifying our graph to account for the distance between the seed CBG, so the algorithm is inclined to cluster closer to the seed CBG.

<u>Distance Graph:</u>

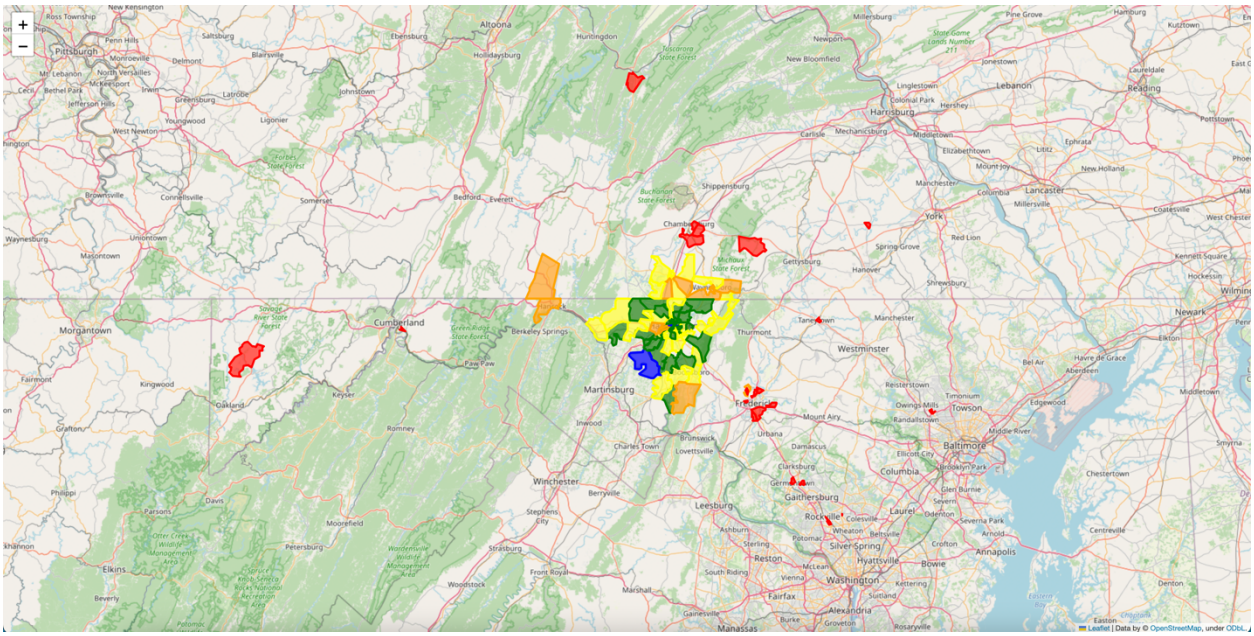
When calculating edge weights (see graph construction), we divided the weight by the distance of the POI to the core CBG. This placed a higher importance on CBGs surrounding the core CBG, and formed better clusters. (with higher ratio)



(Greedy Weight, min population 5000, Ratio = 0.33)

PageRank Nibble:

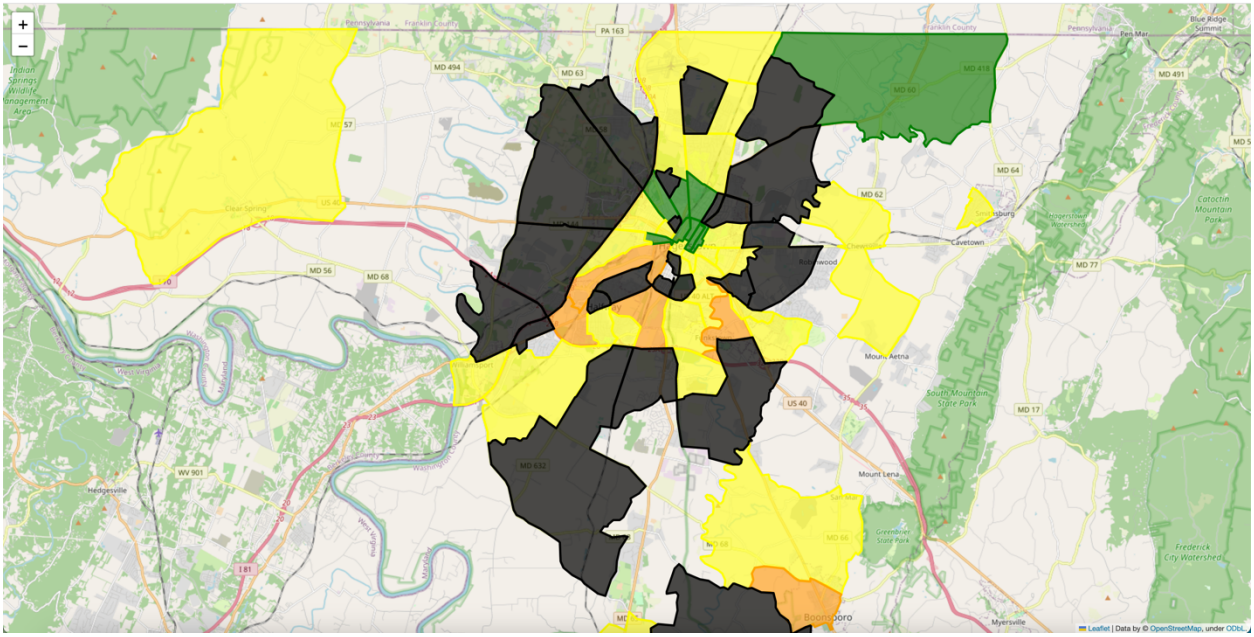
We also tried a known algorithm, PageRank Nibble which uses random walks starting from the seed CBG and identifies 'important' CBGs to include in S. (1-alpha) is the chance that when jumping to a node on a walk, it resets at the seed CBG. There is no minimum population to specify.



(PageRank Nibble, alpha=0.6 (population = 7755), Ratio = 0.32)

Fixing ‘Holes’ in the Cluster

Sometimes S will completely surround a CBG that’s not part of S. For the sake of homogeneity, we attempted to include them by creating an algorithm that counts the amount of shared borders the CBGs of S have with their not included in S neighbors. If more than 3 borders were shared, the CBG was included. While this did include the surrounded CBGs, it also included other CBGs that weren’t surrounded. This algorithm is still unfinished.



(Black CBGs are the ones the algorithm included)