How Far Away: Group VII Report

Deepti Jindal
dg2124@columbia.edu

Andrey Kutser
avk2001@columbia.edu

Zijian Zhou
zz2107@columbia.edu

Introduction

Map generation is a challenging task that requires the map-maker to convey the maximum possible amount of information in a limited amount of space. Due to abundance of information that could possibly be represented, the map-maker must make a choice as to which information is important, thus introducing a level of abstraction that is necessary, given that there is a limit to how much of real world information can be represented on a finite piece of paper. Constrained by the size of a page, the shades of color that can be visually distinguished by a wide range of users and the underlying density of the information that needs to be represented, the final visual representation is not likely to look realistic. However, a good map will have enough information as to allow a user to navigate using the abstract symbols that map to real world objects. Conventional maps, overlaid on top of a flat geographical representation of the land, only include distance information, allowing its user to find a source and destination, and then determine a shortest path that links them, possibly including the size of the road as an indication of its maximum speed. However, the shortest path is not necessarily the fastest one.

Problem

In this project, we explored the possibility of a different map which allows the user to find the fastest path between two points. We tried various approaches in solving this problem, and through experimentation we achieved what we felt was the best possibly map given an additional time constraint imposed on us due to the nature of the project.

The problem of creating a map that reflects the shortest time-distance to travel from any origin to any destination boils down to the problem of drawing all pairs shortest paths of a graph. There has been much research on drawing shortest paths, but most of the research was done on how to draw vertices and edges such that the edges overlap as little as possible; none was focused on usability, meaning how easy is it for the user to pick out the route to take in order to start from any point to any other point in the graph.

We tried to exploit various parameters to maximize usability, visibility, location of the underlying geometry, and levels of detail to ensure that the shortest path between two points can be easily identified.

Map

We tested our players with a map of 50 major cities in the US with their positions given in latitude and longitude coordinates. Their links are of different speeds, links can be country roads, highways and even airline routes.

Baseline Strategy

Internal Representation

Since we are given a set of points and links which connect them, we felt that a graph data structure would be the most reasonable internal representation of our data. This allowed us to perform various graph theory algorithms on our data, and allowed us to focus on the visual representation of the map.

All Pairs Shortest Path

We used Dijkstra's Algorithm to compute the quickest path between two cities time taken to traverse the link as the edge weight, and ran the algorithm over all points to compute the All Pairs Shortest Path result. While running this algorithm, we were able to compute some statistics based on the data, such as which cities have the most incoming links, which edges are in shortest paths and also the mean and standard deviation of this data, providing us with information which we could then use to highlight the importance of some parts of the map, and diminish others. Links not on any quickest paths are discarded when displayed.

Visual Representation

We draw a map as a graph, with straight edges representing roads or links between points on the map, and square bullets representing the locations. Links not on any quickest paths are discarded, since they would only serve to confuse the user by giving him or her suboptimal choices. We include all the cities on the map, regardless of the number of links coming in and out of them.

Distortion

The map is scaled to the size of the window so that it takes full advantage of the space. By scaling the map to the size of the window, we are introducing some distortion into the map. The map will be distorted based on the difference in aspect ratio of the viewing window and the original map. However, since this is a linear distortion, relative horizontal and relative vertical distances will remain the same, with only angles between points changing. We felt that the trade off between the gain space gain versus minor distortion was justified, keeping in mind that due to the underlying space constraint, this type of distortion is almost implicit.

Reduction of Label Clutter

We defined "label clutter" as overlapping textual labels. We de-clutter them by moving the actual points in a radial fashion, using the point being overlapped as the origin. This technique results in points being moved around somewhat, further increasing the distortion of the map, possibly introducing relative distortion of points, making them geographically inaccurate. This did not matter in our experiments that did not rely on the geography, which the user might be familiar with, however we found that the type of distortion it introduced had been negligible, since seeing a label is more important. This de-cluttering did not always give the optimal de-cluttering in extremely cluttered portions of the map, but in general it performed well and had an added aesthetic benefit of a staircase-like arrangement of points and labels.

Strategy Analysis

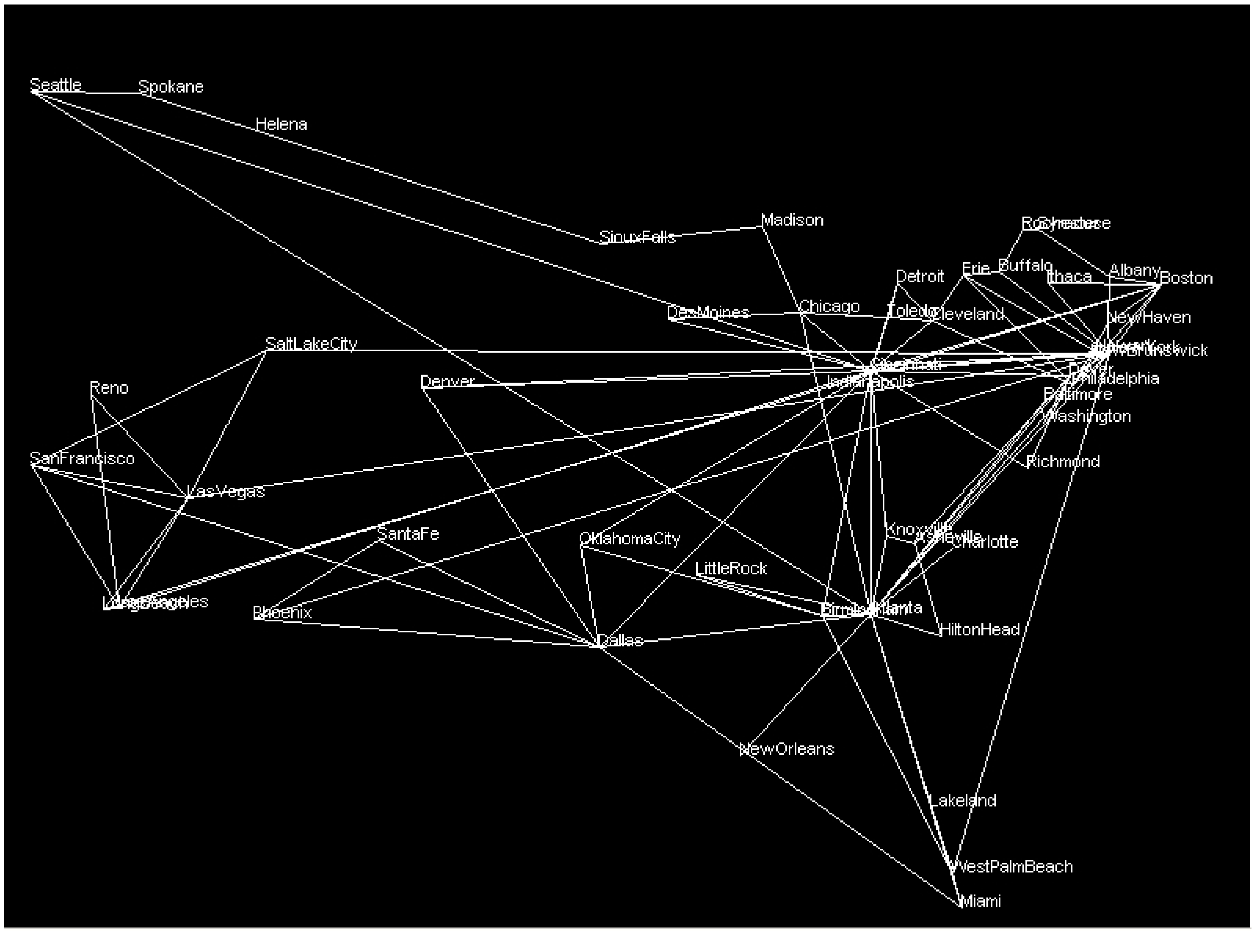


Figure 1: All Links

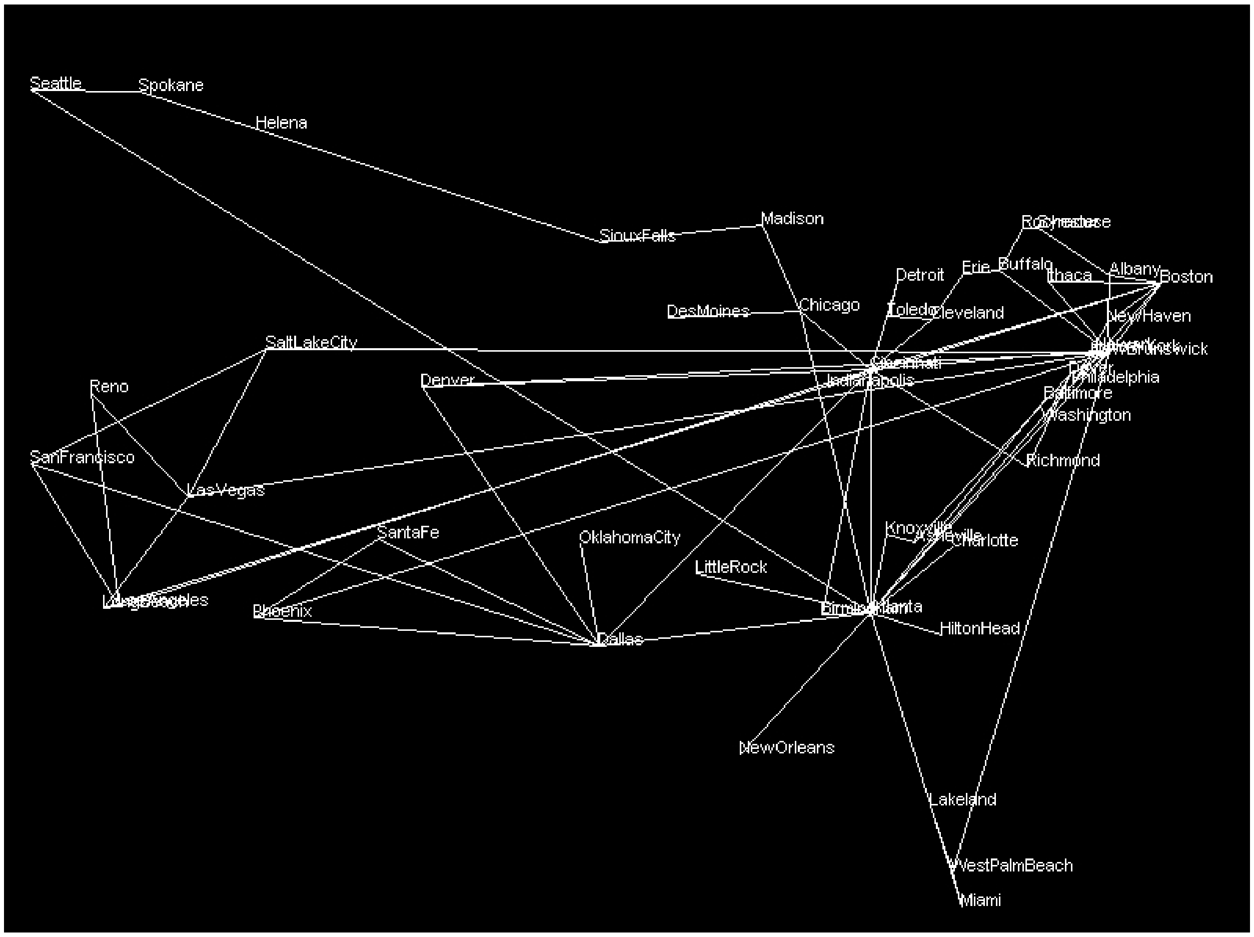


Figure 2: Dijkstra’s Algorithm

After implementing Dijkstra’s Algorithm we removed all edges not contained in any of the shortest paths.

The scaling and label de-cluttering worked relatively well and cleared the layout of the cities on the map. This can be noticed in the later displays of each player.

Player Analysis

Map 1: Rainbow Ribbons

Motivation

The motivation behind Rainbow Ribbons is to allow the user to view the shortest path from any origin to any given destination. By looking at Figure 2, it is not easy to be able to see which path is the shortest. In order to make this process easier the idea is that the map displays each path in a different color, hoping the user can follow the color of the path to their desired destination. Since there are so many links to be displayed if each path is spread out then too much space would be taken up in the map. Thus, it was decided to display all the links bundled together so that the user can follow the route to their ultimate destination by following the color.

Strategy

When we are drawing the links on the map, each path will be assigned a color and every time path shares links, the shared links will be drawn separately in a bundled fashion, as in Figure 3.

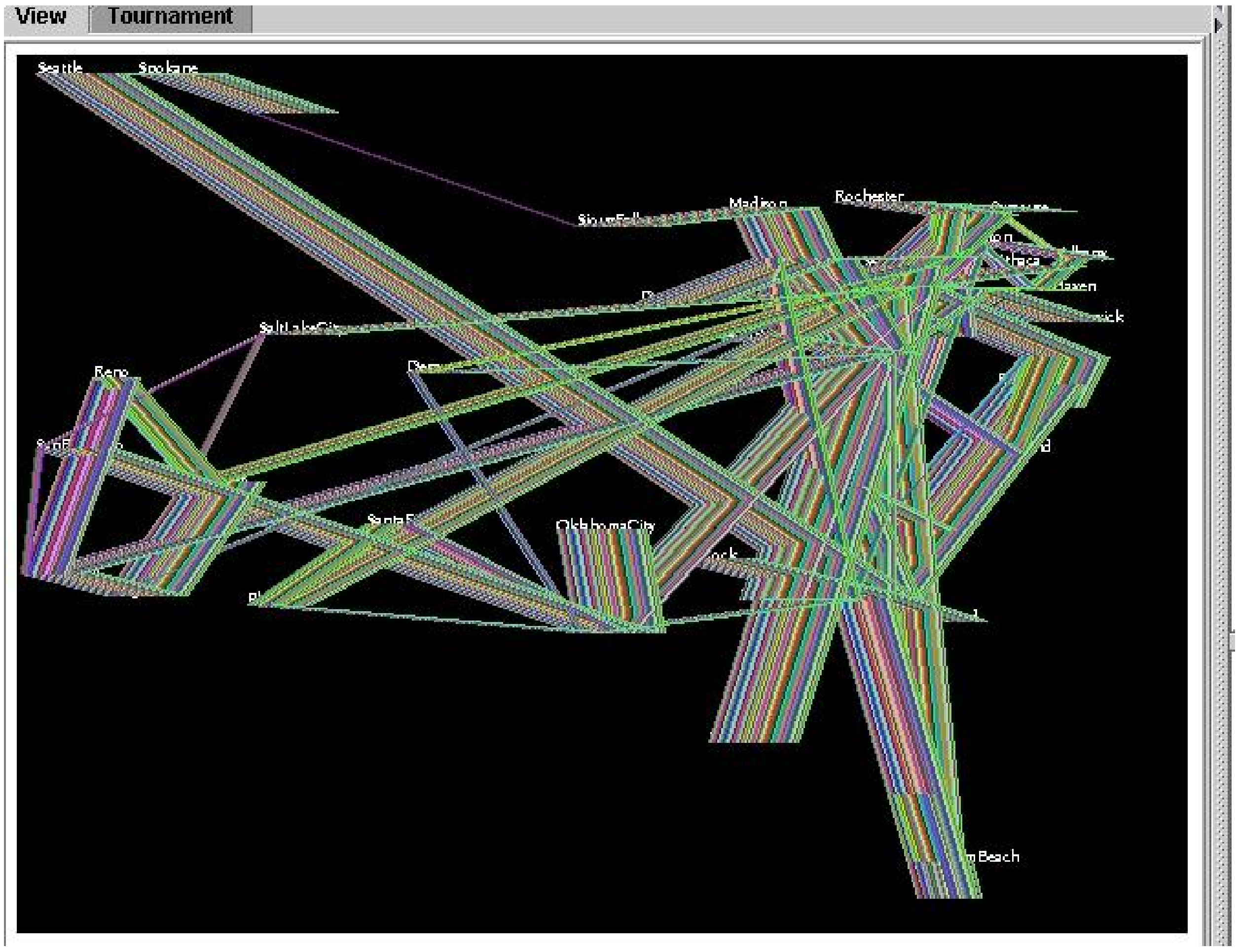


Figure 3: Player Rainbow Ribbons

Analysis

While this approach should theoretically allow the user to unambiguously trace their routes, however, the number of paths bundled made the links too thick. This concept resulted in the task of path finding very difficult. This approach gave us incite on a displaying mechanism. By keeping track how important the link is, we can somehow point out the important links on the map via the thickness of the link.

Map 2: Trick or Treat

Motivation

The motivation behind Trick or Treat is to provide the user as much information using different mechanisms and still keep the geography relatively accurate. The mechanisms that are explored are color, thickness of lines, and luminosity of edges.

Strategy

Color and Stroke

The choice of color and line stroke was very difficult and also very important in generating a good map. Even though we were told that a map that "looks" good is not necessarily a good map, we understood that a good map might must also be aesthetically pleasing and not strain its user by bad color choices or inappropriate font sizes. Though the concept of beauty is subjective, we tried to use our best judgment in choosing the visual components of the map, keeping in mind people who are insensitive to differences between various colors.

Hubs

Points with a greater number of incoming quickest path edges are made more prominent by the use of a larger square for representing the location. This is done in conventional maps as well, and we felt it was a good paradigm. However, the label font size remains the same as it introduces extra clutter without adding must information to the map.

Time

Links are colored based on the length of time it takes to traverse the link. We chose only 3 colors to determine the speed rating - blue for "slow", green for "medium" and red for "fast". The relative speed of the links is split into these three categories by normalizing the time of the link using the normal distribution and shifting the value based on the mean and standard deviation over all the time values. The normal curve is then split into three parts, each containing approximately 33% of the points. "Fast" will coincide with the values more than half a standard deviation from the mean, "medium" is values within half a deviation from the mean, and "slow" is all remaining values. This ensures a fairly even division of colors, assuming that there are enough values to sample so that the distribution is approximately normal. Our choice of color was mostly subjective. We tried to keep it such that it would be relatively easy for the user to figure out which links are for faster speeds and which are for slower without having to look at the legend. We felt that red tended to draw more attention, so it appears to be faster, while green and blue colors tend to draw less attention to themselves and are paler in comparison. In addition to having a color coded time rating, we also felt that including the actual time a link takes will allow a user who is perhaps not a “visual” person to simply add the path edge values and achieve the fastest path that way. However, since the time labels are additional and, for the most part unnecessary information, we drew the edge labels in a paler color and smaller font, as seen in Figure 9. This way, they will not compete with edges and cities for visual attention, while still being discernable if a user’s need to scan them should arise.

Popularity

Not all links on the map are equally important. Some links are a part of multiple shortest paths, and thus their popularity decides on how prominent they will be in the graphical representation. We used line thickness to represent a link's popularity, whose thickness ranges from 5 (most popular) to 1 (least popular), determined by the formula below.

$$\text{THICKNESS} = \text{MAX}\,[5 * (Z_{pop} - Z_{popMin}) / (Z_{popMax} - Z_{popMin}),\ 1]$$

Z_{pop} is the normalized popularity of the given link, Z_{popMin} the normalized minimum popularity and Z_{popMax} the normalized maximum popularity. Thus, a thicker link will most likely be one that the user must traverse for a random choice of a source and destination point, and will have to be more prominent. Additionally, the intensity of the color of the link will also be decreased for links which have a lower popularity, as seen in Figure 8. Link popularity is also normalized using the standard distribution and the mean and deviation statistics to ensure a smooth distribution of intensity, as computed by the below formula, with the variables having the same definition and value as above, which is then scaled appropriately for the time speed coding.

$$\text{INTENSITY} = \text{MIN}\ [(0.2 + (Z_{pop} - Z_{popMin}) / (Z_{popMax} - Z_{popMin})),\ 1.0]$$

This causes links to appear more faded than their more popular counterparts. By changing the intensity, we try to achieve the same effect as removing a less popular link would have, but without the consequence of removing a part of some quickest path. It also makes the map more readable as there appear to be less links on the map, though upon close inspection, faded links can be identified, if they are actually needed. We used a minimum value of 0.2 for intensity, from the above formula, to ensure that links don't disappear completely.

Legend

Due to the fact that different people have differing interpretations of what it means for a link to be of a certain color or thickness, we include a concise legend to rectify this problem, seen in Figure 4. Thus, the user who might mistake a thick link for one that is faster will be made aware of our convention, and hopefully will incorporate this information when using the map. If the colors are still not as indicative for speed the link speed is given for a final derivation of the best route.

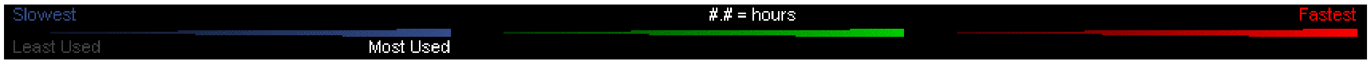


Figure 4: Legend used in final players

Analysis

In Figure 5, we can see how popularity of a link is being implemented using the fading and thickness technique.

We recognized early on in the project that there will be a certain viewing order when a user is faced with a map and is given an origin and destination. Thus, we tried to tailor our color and stroke usage in such a way, as to prioritize things that the user will be required to use, and those that have a minor importance. We understood that initially, a user is look for the origin city, then the destination city. Thus, cities were made to have distinct, bright colors which draw attention to themselves. Then the path links will need to be identified, thus link color, thickness and shading will determine which link the user will likely choose next. Finally, if there is some ambiguity as to the selection of a quickest path, edge labels can be used as a tie-breaker, and so these labels have the lowest priority and most faded colors.

As we can see in the graph below, white city labels were too bright on the black background bringing too much attention to just the city names. In the next two figures different colors were chosen to see the effect with visual appeal. From the two displays it was concluded that fading was a better seen on a black background, which we chose for our final submissions. In this map we can also see the indication of hubs with a bigger bullet.

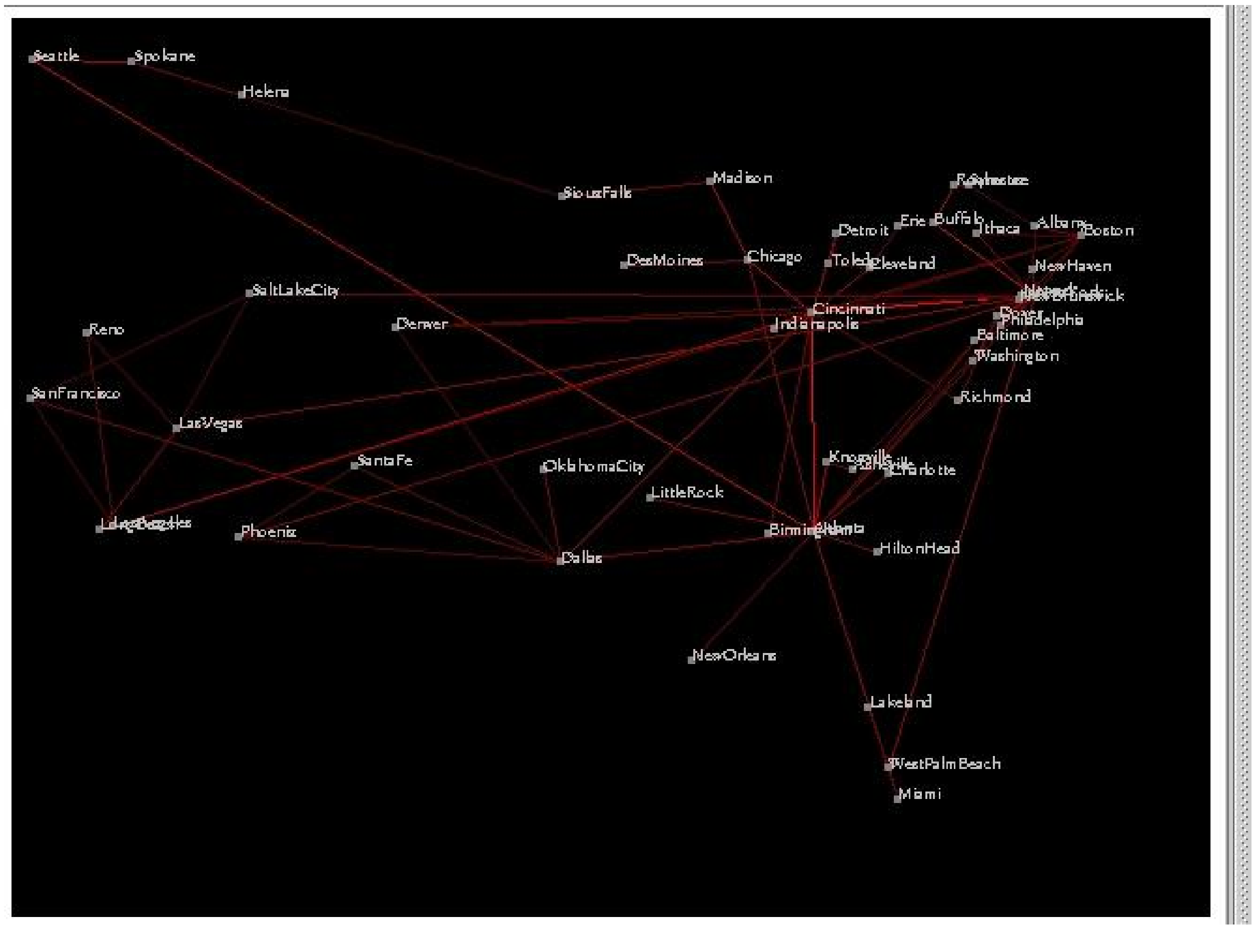


Figure 5: Screen shot of the fading effect

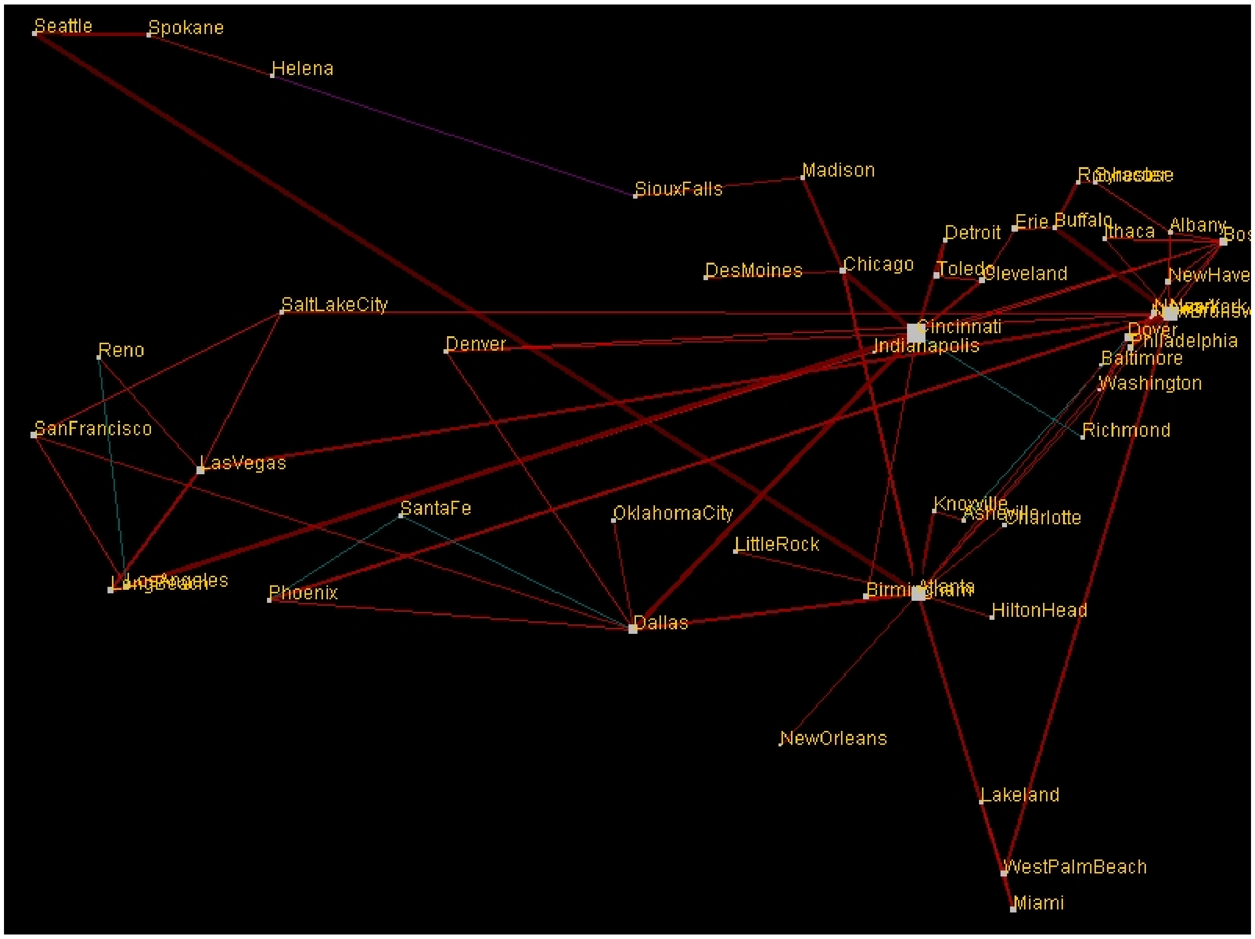


Figure 6: Screen shot of different color choices and addition of Hubs

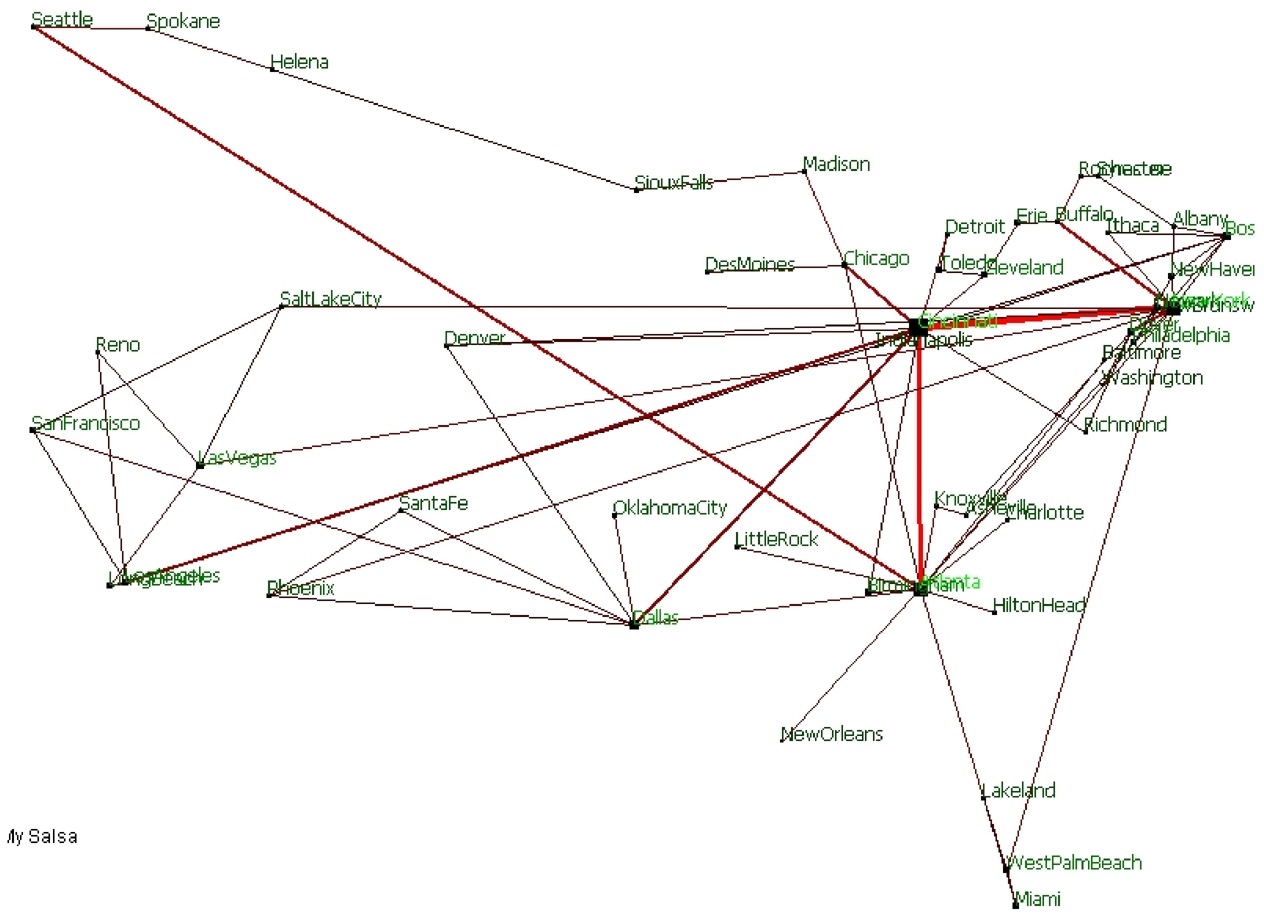


Figure 7: Screen shot of White Background and Thickness is also added for popularity of a link

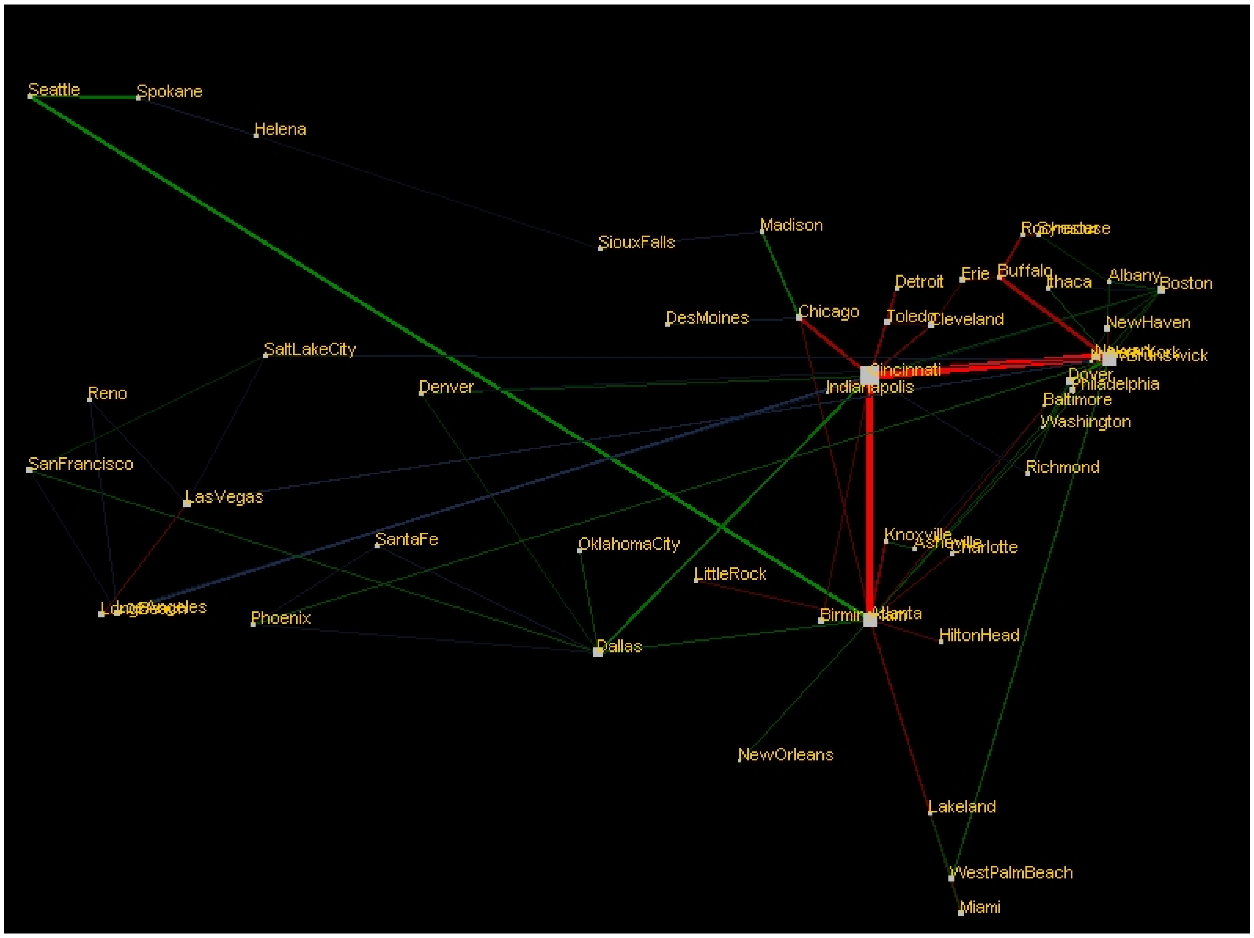


Figure 8: Screen shot of colors

In this figure we can see the coloring of speeds added.

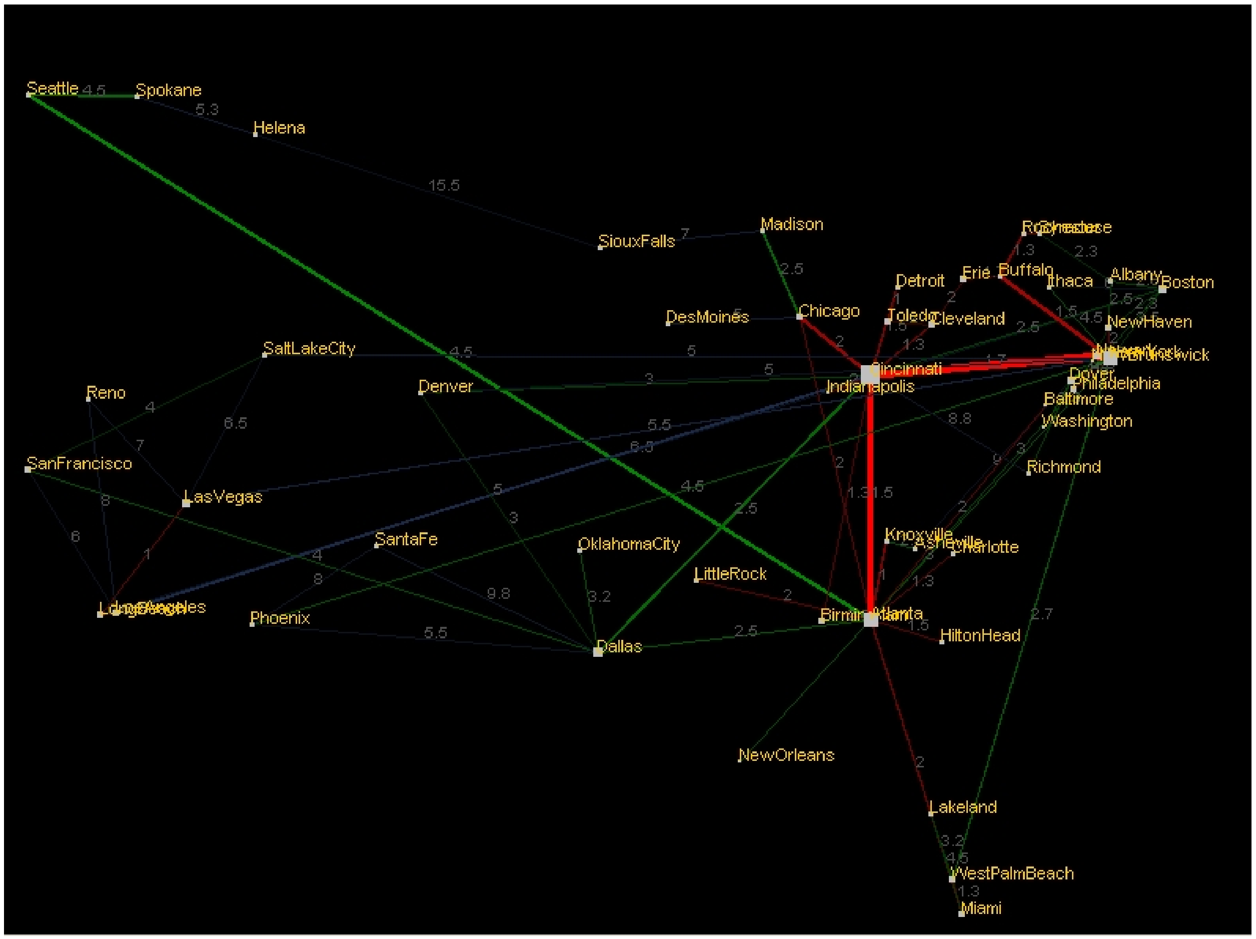


Figure 9: Screen shot of time labels for edges

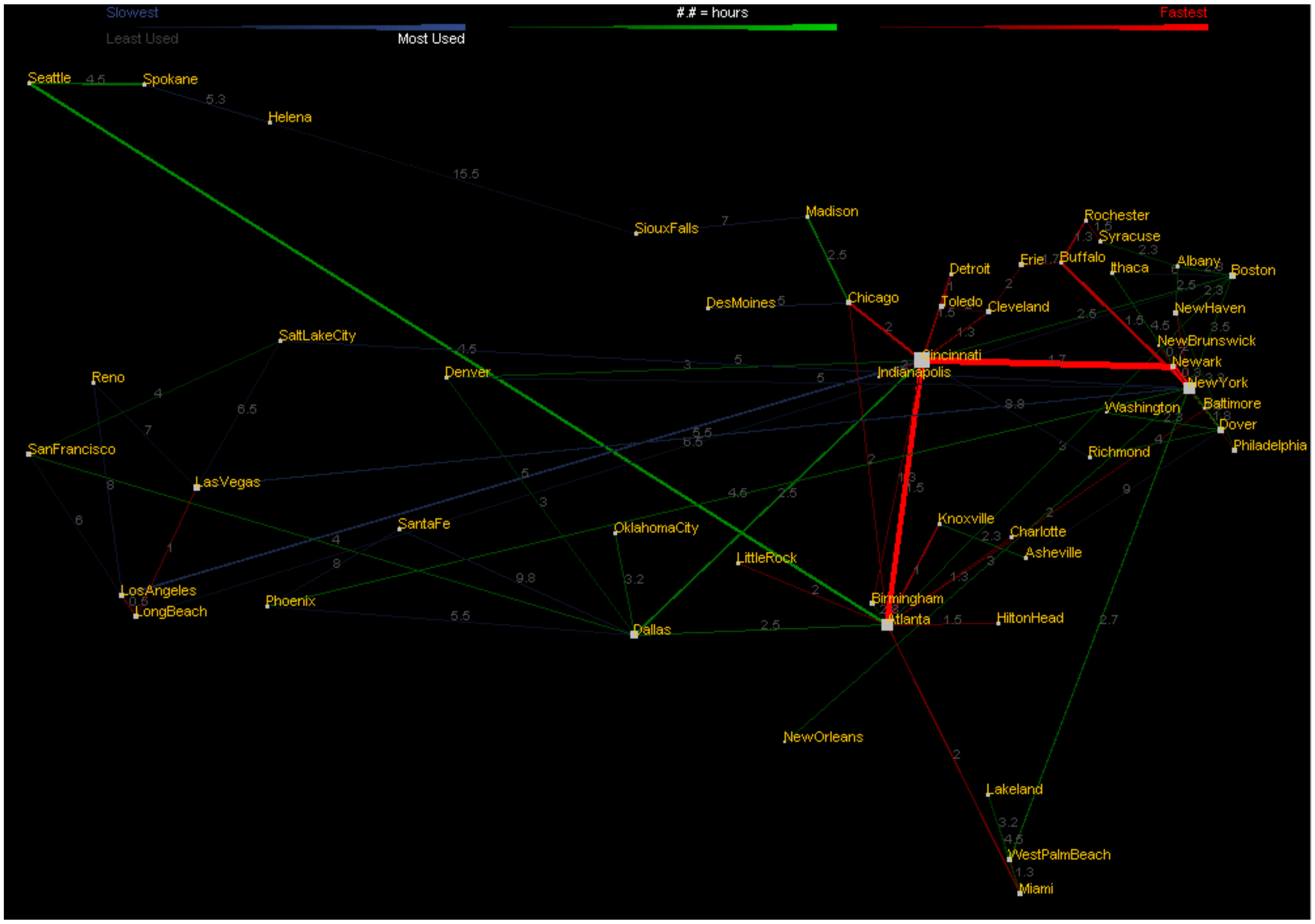


Figure 10: Screen shot of final map with de-clustering

Map 3: Alphabet Soup (Trick or Treat variation)

Motivation

In this player we wanted to go away from a map that was geographically correct. We noticed that when there is a clutter in the northeast section then the map was less readable in the northeast. We try to maximize the area for each city by creating a grid system.

Spring Embedding

In the first attempt the idea of spring embedding was implemented. Spring embedding is a graph displaying technique that is commonly used to space the nodes of a graph based on a set of virtual “springs” being attached to the vertices of a graph which are then simulated using physical principles to cause the tensions and compressions resulting from the spring forces to move the nodes to a position with the minimum amount of energy. This idea seemed good on paper, prompting us to actually implement it. However, having implemented this technique we found it to be extremely unstable and leading to irreproducible results and “explosions” which resulted in points getting stuck in corners and breaking the algorithm. While in general, the output looked like the random gridding image below, we scrapped this idea because it required too much time to execute the simulation, and the results were not even close to optimal. Through this experiment we found that simple heuristic rules were better at performing the task than complicated and possibly unstable methods which don't necessarily guarantee a plausible solution.

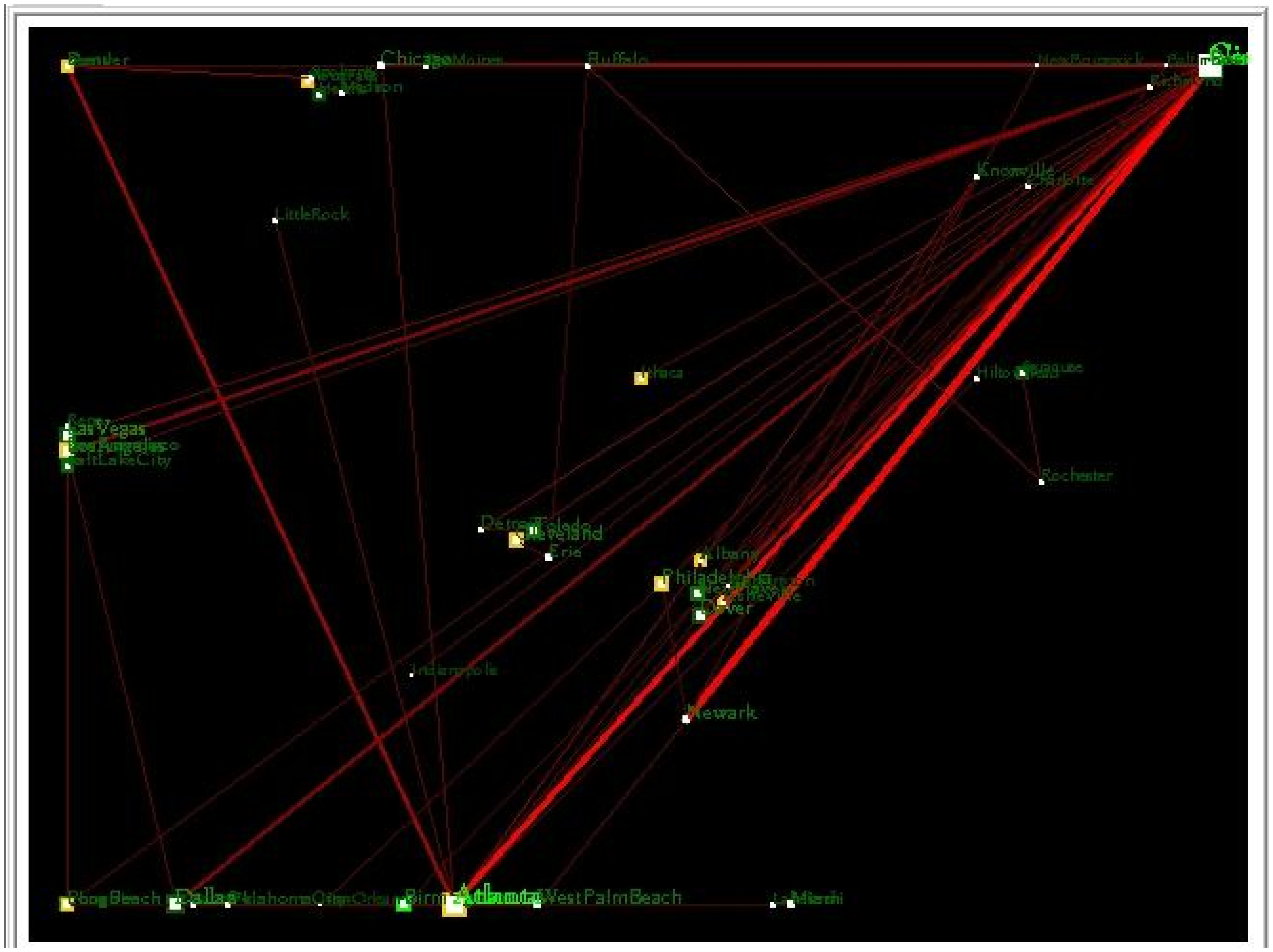


Figure 11: Screen shot of spring embedding.

Gridding

We break up the map into five rows, each with n columns. We calculate the number of columns by the following formula:

```
Columns = (total number of cities)/rows
```

Then we lock in the position of each city so it lines up like a grid. However we noticed in this method the links would overlap, as in Figure 11, not allowing the user to identify which cities go through which link.

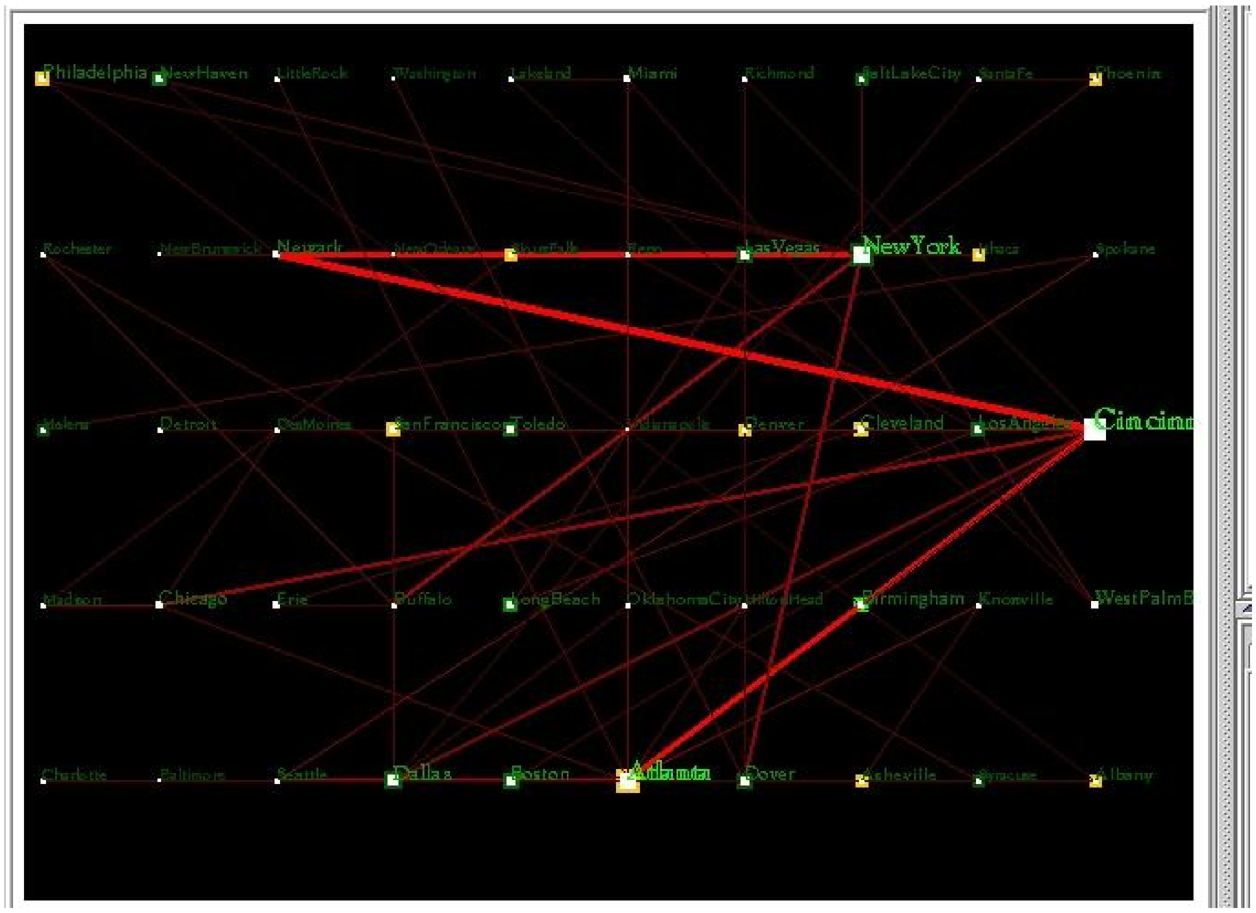


Figure 12: Screen shot of the gridding in action

Random Gridding

We then tried to break the map area into x number of areas, which corresponds to the total number of cities and then randomly place the city into its given area. This fixed the overlap in links noted in the regular gridding strategy, as seen in Figure 13, but we noticed it was hard for a user to find the city they are looking for.

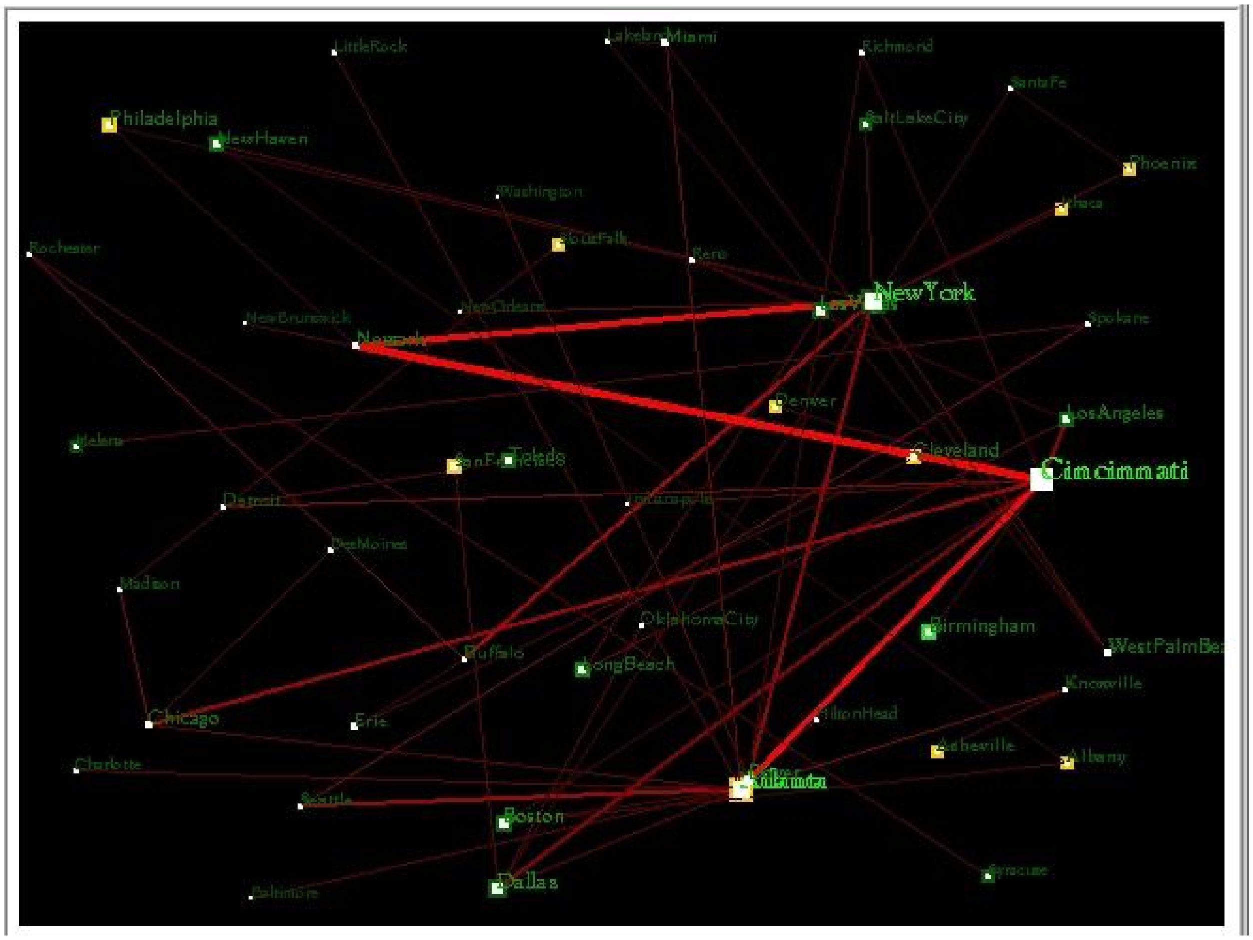


Figure 13: Screen shot of random grid

Alphabetized

The map was then put in alphabetical order to ease the readability process. The other benefit we noticed by putting the map alphabetically then the user can look at an unknown area and can still find a particular city easily.

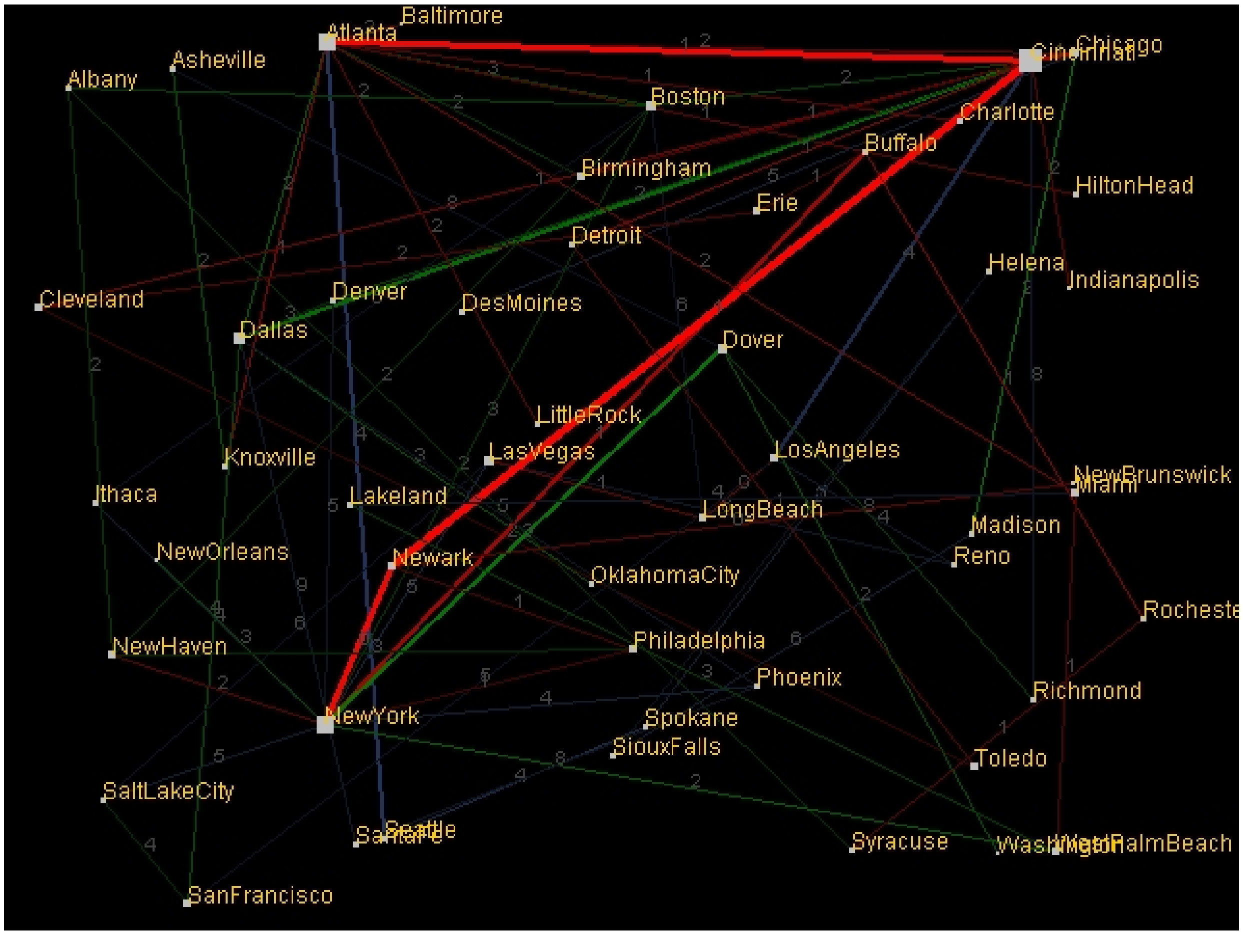


Figure 14: Screen shot of random grid alphabetized integrating with many ideas from Trick or Treat

So the final map looked like this when integrated with the de-clustering.

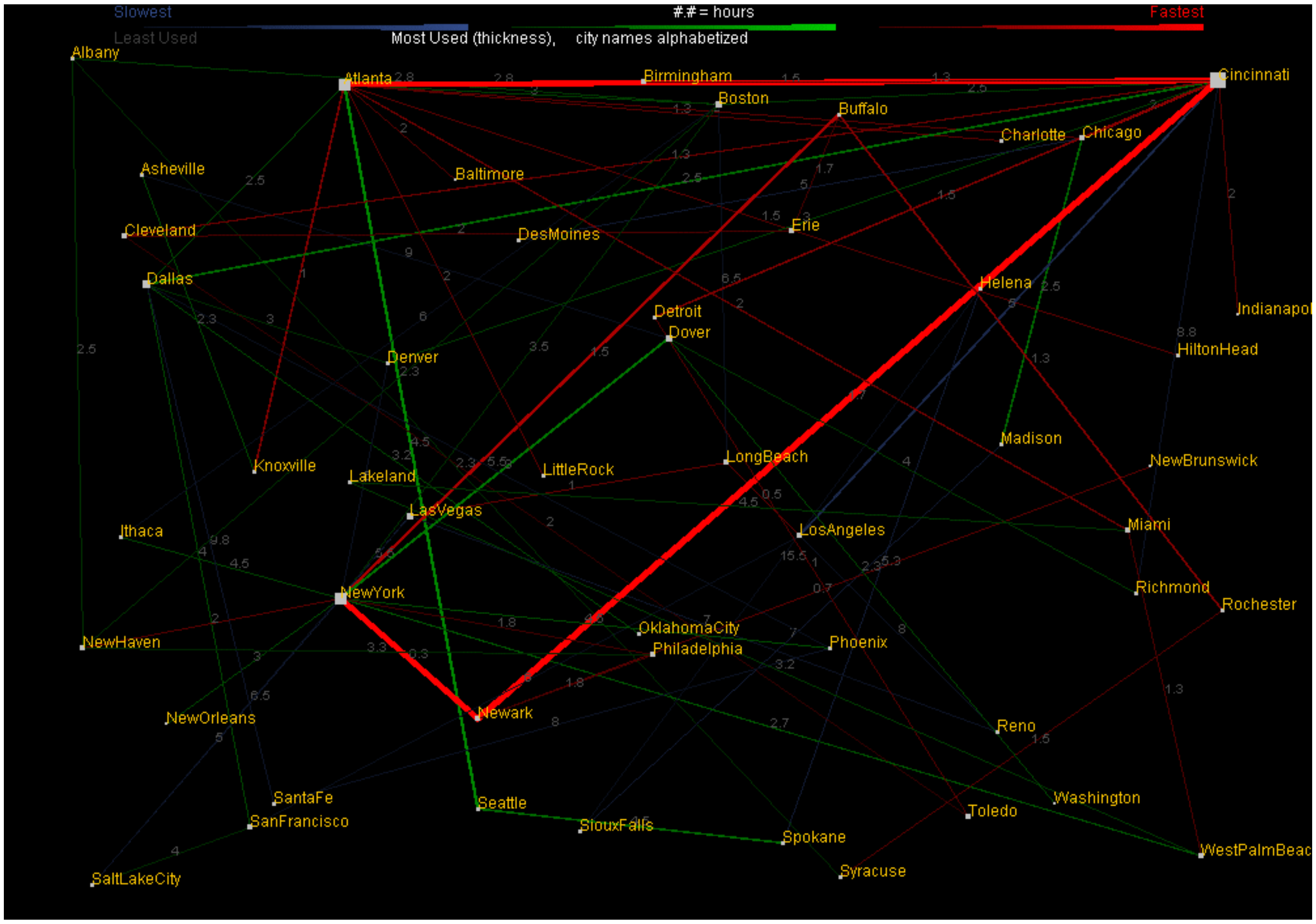


Figure 15: Screen shot of Final Map

Analysis

We noticed from the final map that one of the biggest downfalls of this idea is links are made longer between cities close together. For example geographically New York and Albany are very close together, so there corresponding link will be very short in the geographical map of US. However, in this alphabetized gridding system the link will increase significantly.

Results

Final Submissions

In order of appearance, below:

Figure 16: Trick or Treat with Northeast

Figure 17: Trick or Treat with Australia

Figure 18: Trick or Treat with Subway Map

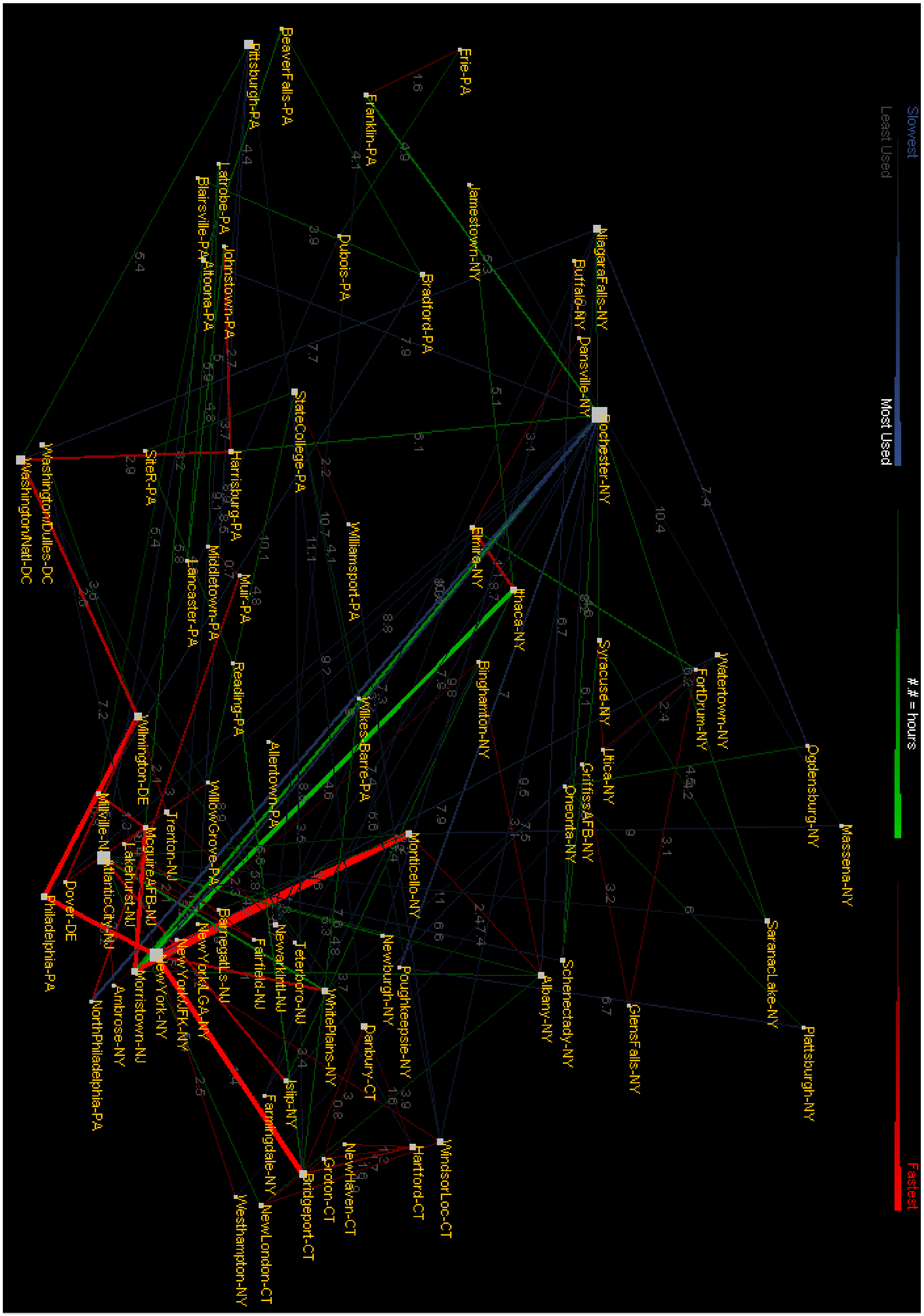
Figure 19: Trick or Treat with US Capitals

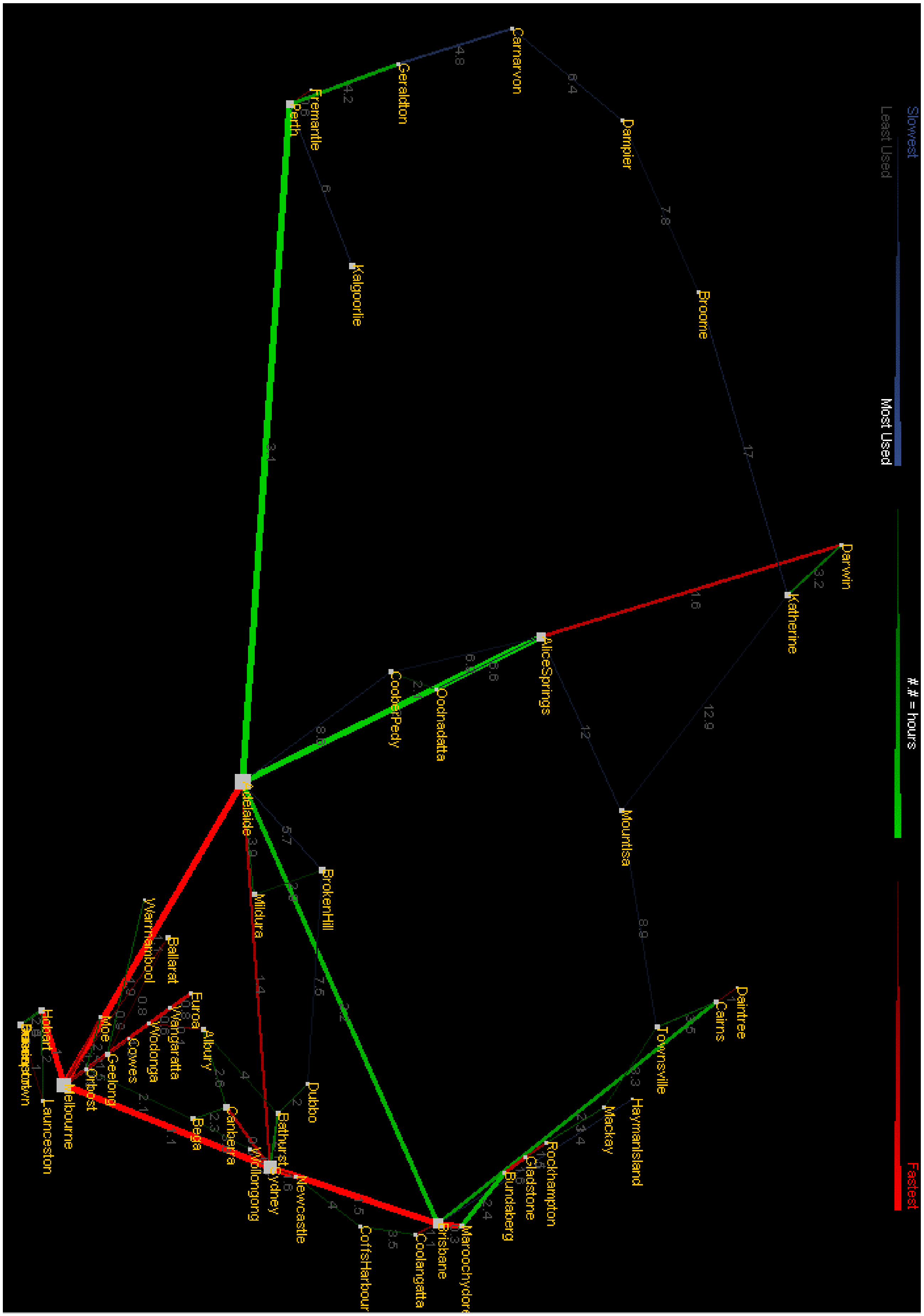
Figure 20: Alphabet Soup with Northeast

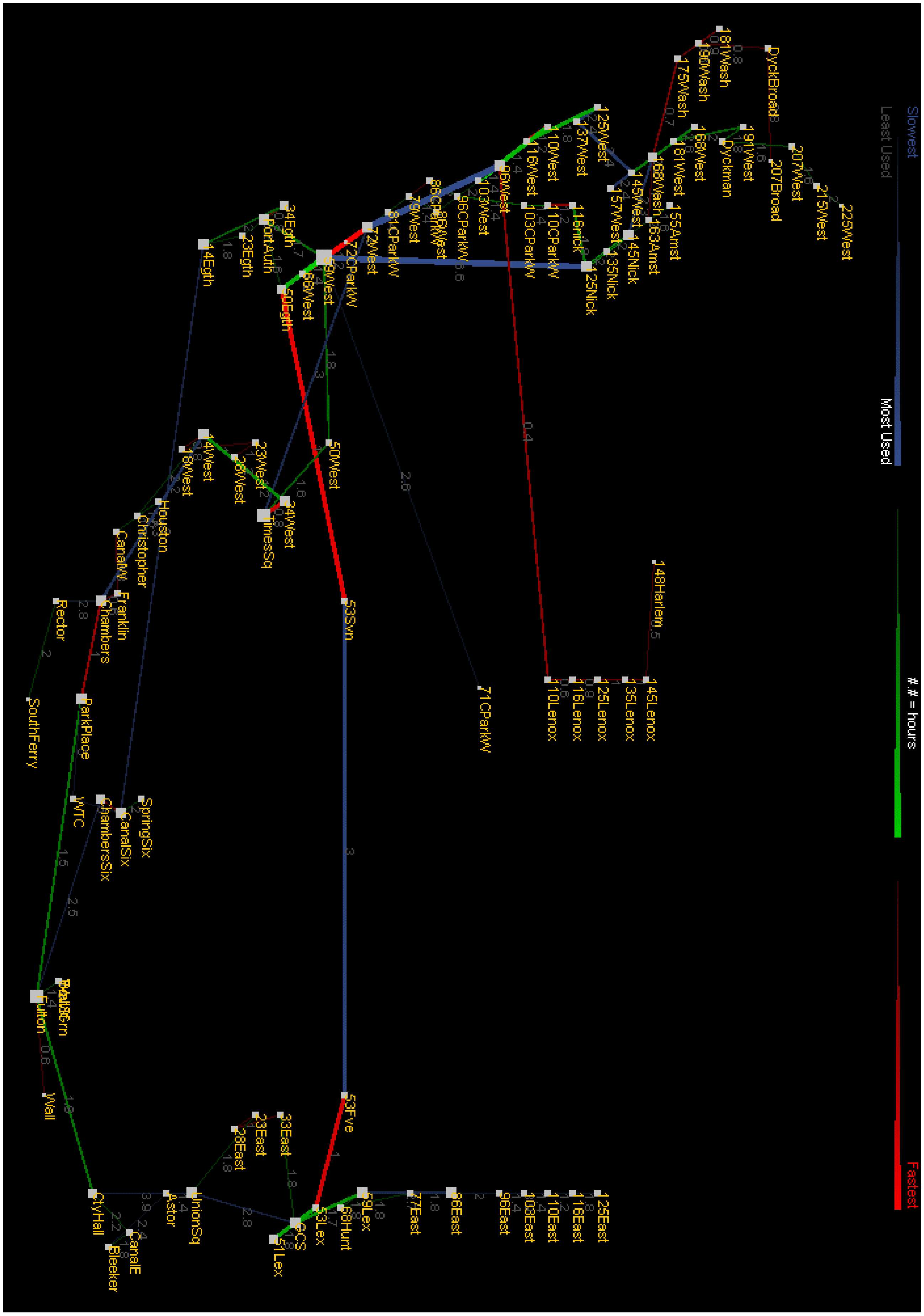
Figure 21: Alphabet Soup with Australia

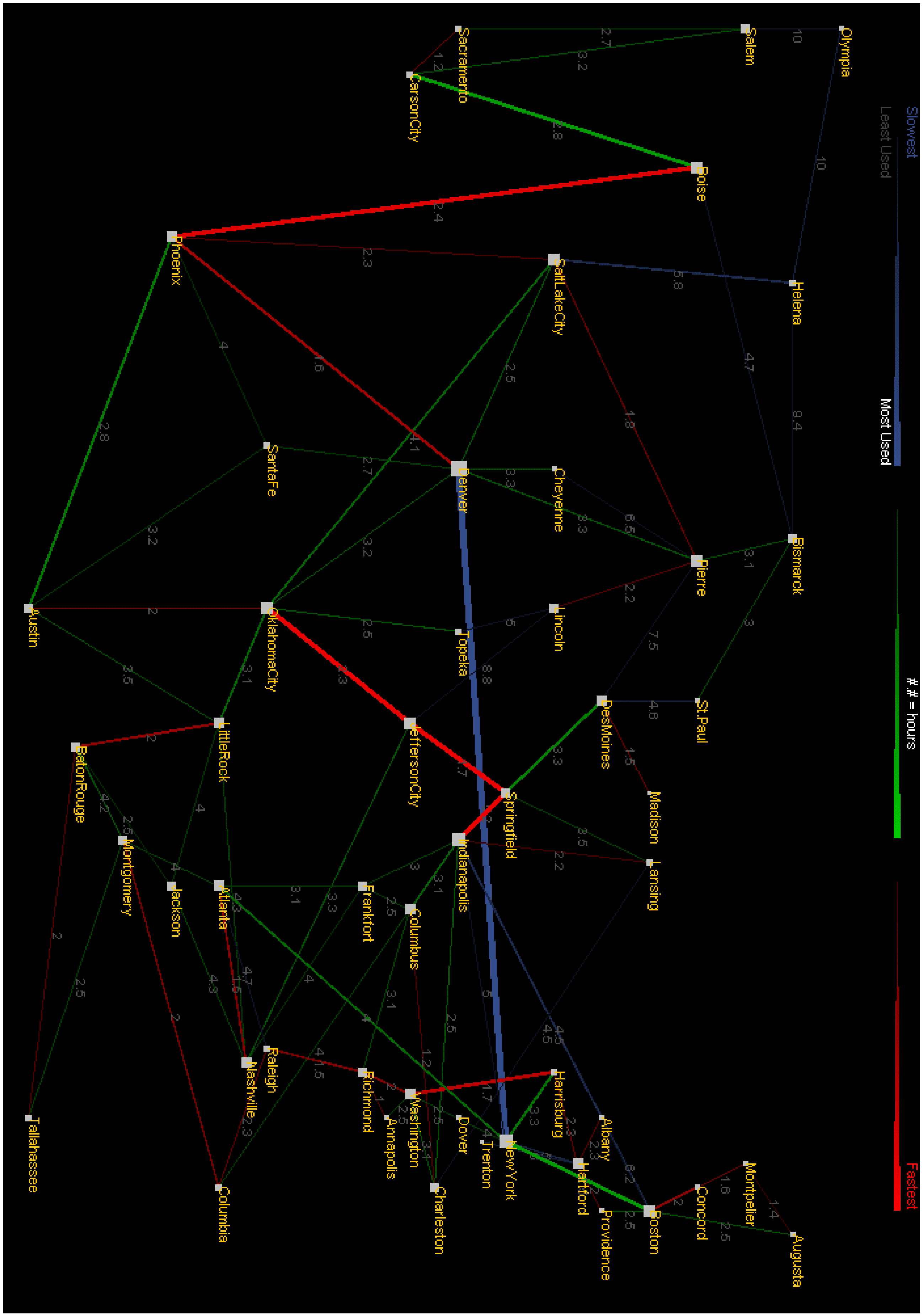
Figure 22: Alphabet Soup with Subways

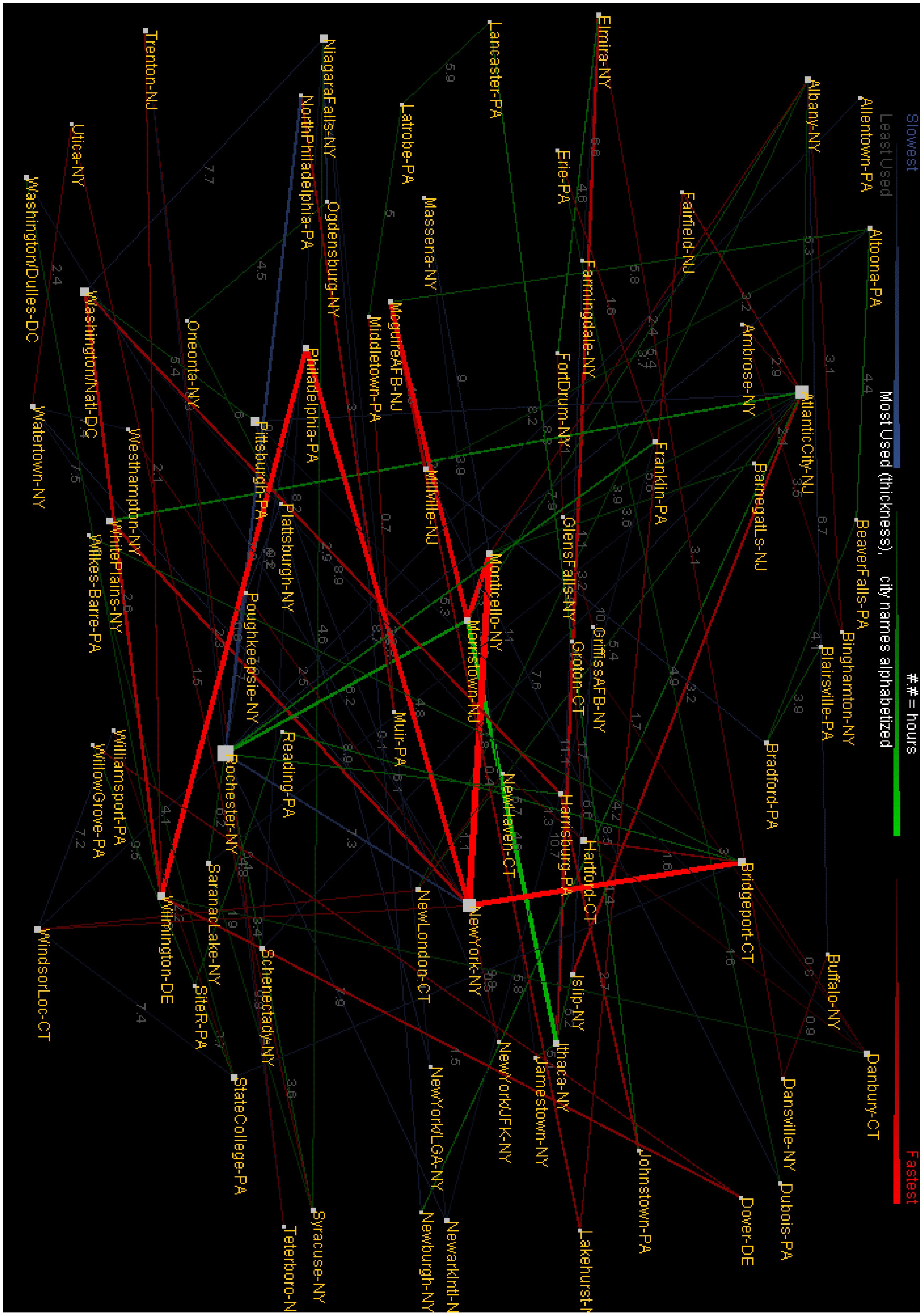
Figure 23: Alphabet Soup with US Capitals

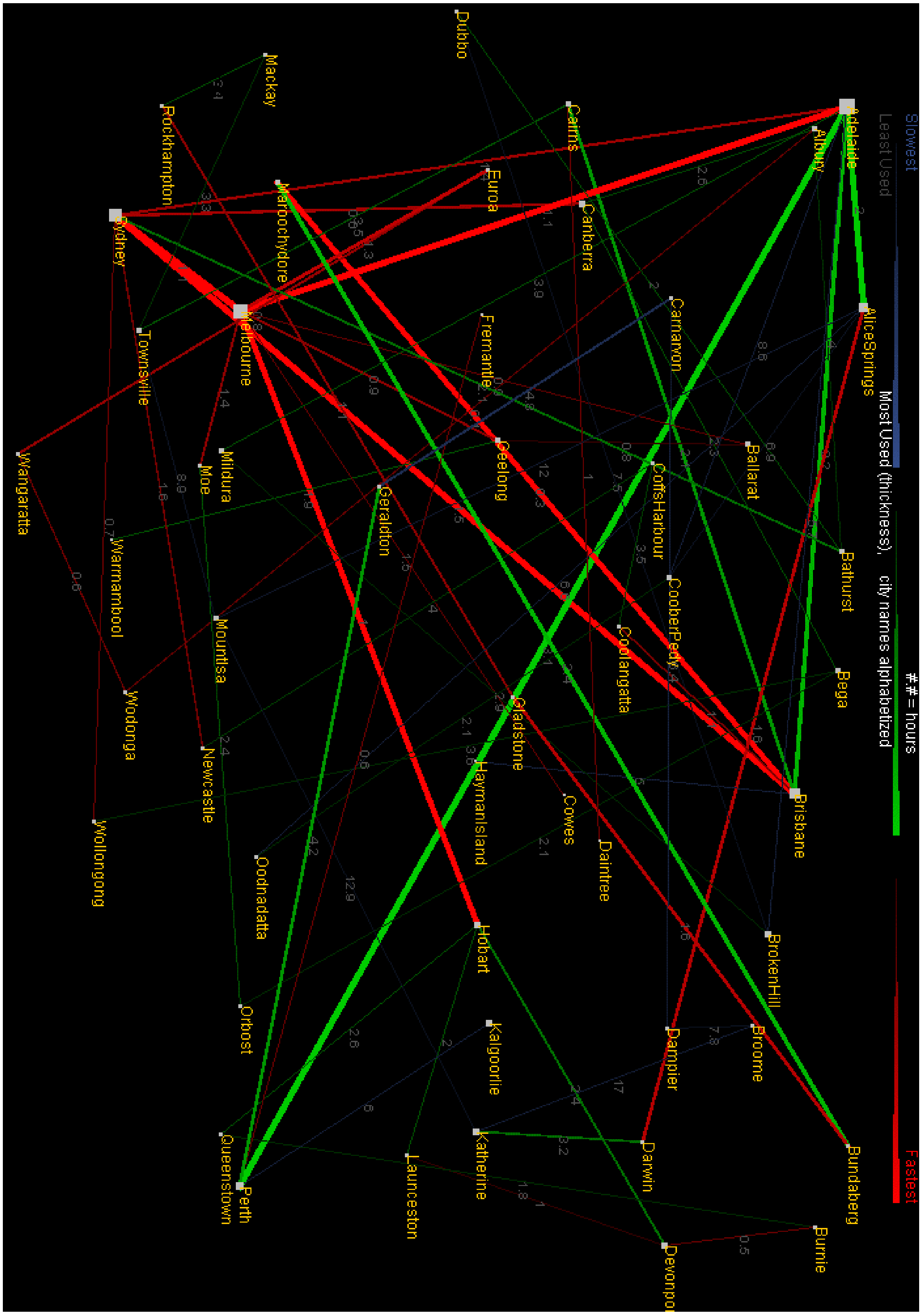

Slowest
Least Used
Most Used
#.# = hours
Fastest
Erie-PA
Franklin-PA
BeaverFalls-PA
Pittsburgh-PA
Latrobe-PA
Blairsville-PA
Johnstown-PA
Altoona-PA
Dubois-PA
Bradford-PA
Jamestown-NY
NiagaraFalls-NY
Buffalo-NY
Dansville-NY
Rochester-NY
StateCollege-PA
Harrisburg-PA
SiteR-PA
WashingtonDulles-DC
WashingtonNatl-DC
Williamsport-PA
Elmira-NY
Ithaca-NY
Middletown-PA
Muir-PA
Lancaster-PA
Reading-PA
Syracuse-NY
Watertown-NY
FortDrum-NY
Binghamton-NY
WilkesBarre-PA
Utica-NY
GriffissAFB-NY
Oneonta-NY
Ogdensburg-NY
Massena-NY
Wilmington-DE
Allentown-PA
WillowGrove-PA
Trenton-NJ
Millville-NJ
McguireAFB-NJ
Lakehurst-NJ
AtlanticCity-NJ
Dover-DE
Philadelphia-PA
Monticello-NY
BarnegatLs-NJ
NewYorkLGA-NY
Fairfield-NJ
NewarkIntl-NJ
Teterboro-NJ
Newburgh-NY
Poughkeepsie-NY
Albany-NY
Schenectady-NY
SaranacLake-NY
GlensFalls-NY
Plattsburgh-NY
NewYork-NY
NewYorkJFK-NY
Morristown-NJ
Ambrose-NY
NorthPhiladelphia-PA
WhitePlains-NY
Danbury-CT
Islip-NY
Farmingdale-NY
Bridgeport-CT
Groton-CT
NewHaven-CT
Hartford-CT
WindsorLoc-CT
Westhampton-NY
NewLondon-CT

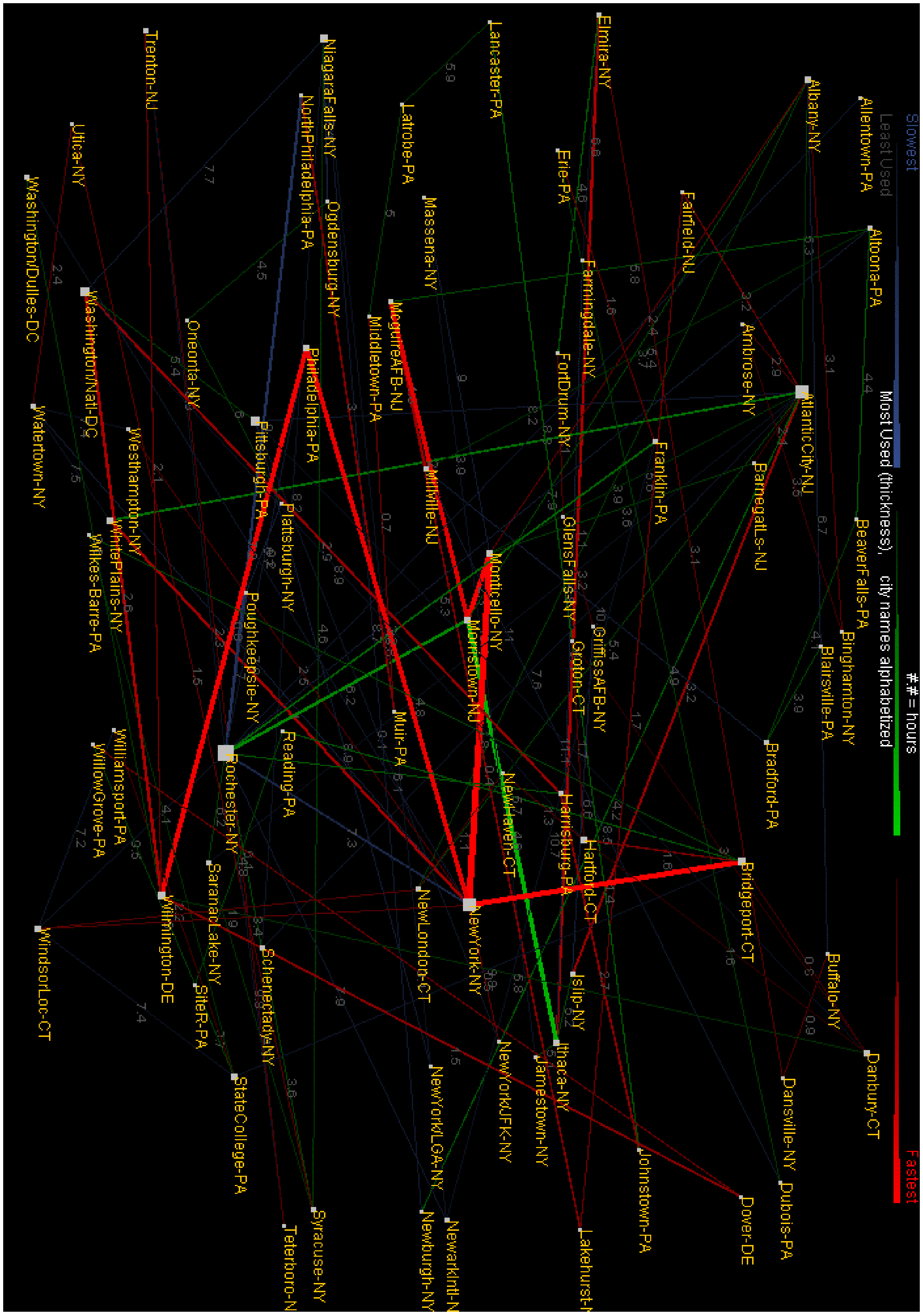

Slowest
Least Used
Most Used
# = hours
Fastest
Carnarvon
Dampier
Geraldton
Fremantle
Perth
Kalgoorlie
Broome
Darwin
Katherine
AliceSprings
CooberPedy
Oodnadatta
MountIsa
Adelaide
BrokenHill
Mildura
Warrnambool
Ballarat
Euroa
Wangaratta
Wodonga
Albury
Cowes
Geelong
Moe
Orbost
Melbourne
Hobart
Launceston
Bega
Canberra
Dubbo
Bathurst
Sydney
Wollongong
Newcastle
CoffsHarbour
Coolangatta
Brisbane
Maroochydore
Bundaberg
Gladstone
Rockhampton
Mackay
HaymanIsland
Townsville
Cairns
Daintree
4.8
6.4
4.2
6
7.8
17
3.2
1.6
12.9
12
8.9
3.1
5.7
3.9
1.4
7.5
2.4
3.5
1.5
4
2

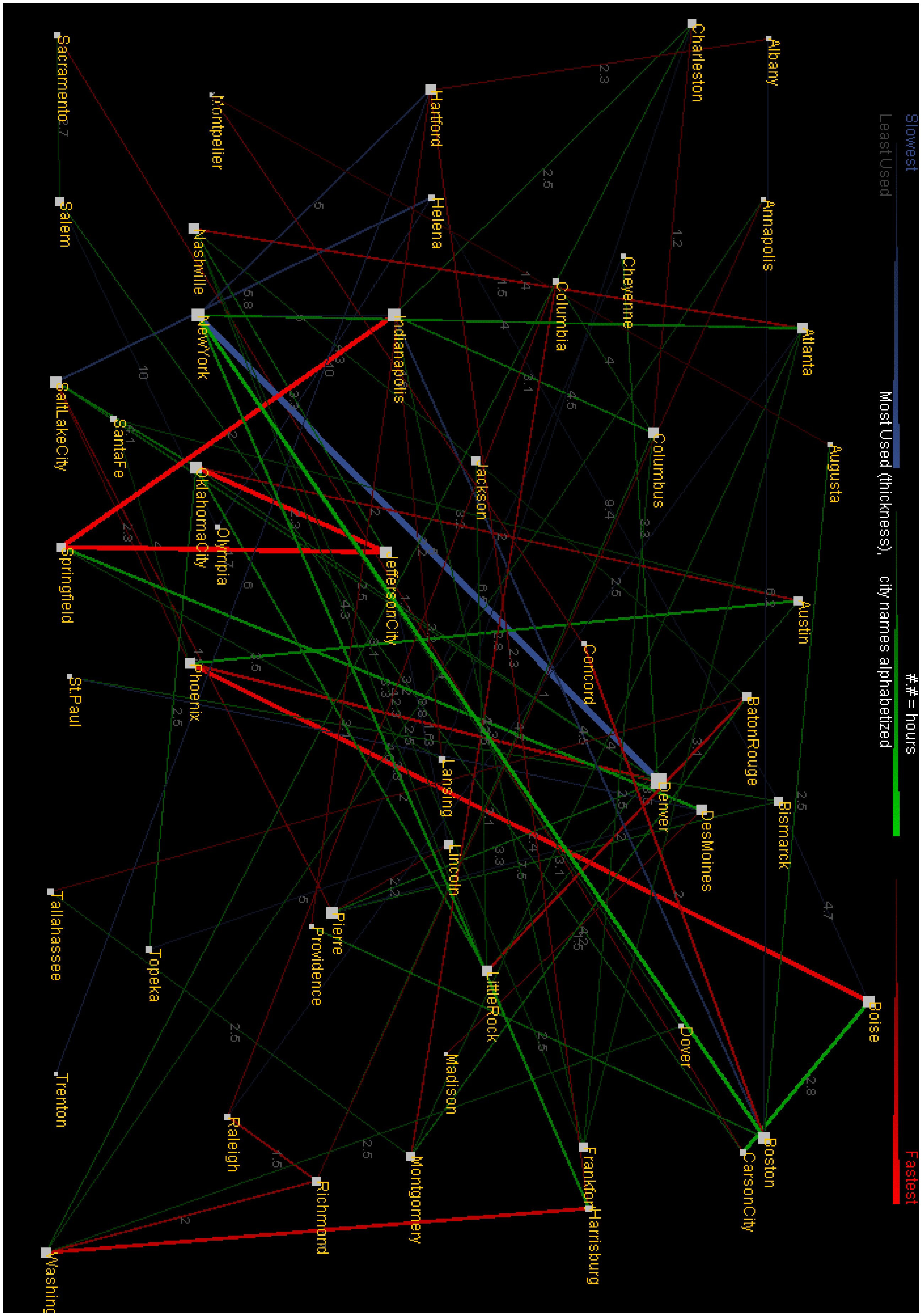

Slowest
Least Used
Most Used
#.# = hours
Fastest
181Wash
190Wash
175Wash
DyckBroad
191West
168West
Dyckman
207West
207Broad
215West
225West
181West
125West
137West
110West
116West
96West
103West
86CParkW
81CParkW
79West
96CParkW
103CParkW
110CParkW
116Nick
125Nick
135Nick
145Nick
157West
145West
168Wash
163Amst
155Amst
72CParkW
72West
59West
66West
50Egth
34Egth
PortAuth
23Egth
14Egth
50West
34West
TimesSq
23West
28West
14West
18West
Houston
Christopher
CanalW
Franklin
Chambers
Rector
ParkPlace
SouthFerry
WTC
ChambersSix
CanalSix
SpringSix
Fulton
Wall
CtyHall
Bleeker
CanalE
Astor
UnionSq
28East
23East
33East
GCS
51Lex
53Lex
53Fve
53Svn
68Hunt
59Lex
77East
86East
96East
103East
110East
116East
125East
71CParkW
110Lenox
116Lenox
125Lenox
135Lenox
145Lenox
148Harlem


Slowest
Least Used
Most Used
# = hours
Fastest
Olympia
Salem
Sacramento
CarsonCity
Boise
Phoenix
SaltLakeCity
Helena
SantaFe
Denver
Cheyenne
Bismarck
Pierre
Austin
OklahomaCity
Topeka
Lincoln
LittleRock
JeffersonCity
DesMoines
St.Paul
BatonRouge
Madison
Springfield
Montgomery
Jackson
Atlanta
Frankfort
Indianapolis
Lansing
Columbus
Nashville
Raleigh
Richmond
Washington
Annapolis
Harrisburg
Dover
Trenton
NewYork
Albany
Hartford
Providence
Boston
Concord
Montpelier
Augusta
Tallahassee
Columbia
Charleston


Slowest
Least Used
Most Used (thickness), city names alphabetized
#.# = hours
Fastest
Albany-NY
Allentown-PA
Altoona-PA
Ambrose-NY
AtlanticCity-NJ
BarnegatLs-NJ
BeaverFalls-PA
Binghamton-NY
Blairsville-PA
Bradford-PA
Bridgeport-CT
Buffalo-NY
Danbury-CT
Dansville-NY
Dover-DE
Dubois-PA
Elmira-NY
Erie-PA
Fairfield-NJ
Farmingdale-NY
FortDrum-NY
Franklin-PA
GlensFalls-NY
GriffissAFB-NY
Groton-CT
Harrisburg-PA
Hartford-CT
Islip-NY
Ithaca-NY
Jamestown-NY
Johnstown-PA
Lakehurst-N
Lancaster-PA
Latrobe-PA
Massena-NY
MaguireAFB-NJ
Middletown-PA
Millville-NJ
Monticello-NY
Morristown-NJ
Muir-PA
NewHaven-CT
NewLondon-CT
NewYork-NY
NewYorkJFK-NY
NewYorkLGA-NY
Newburgh-NY
NewarkIntl-N
NiagaraFalls-NY
NorthPhiladelphia-PA
Ogdensburg-NY
Oneonta-NY
Philadelphia-PA
Pittsburgh-PA
Plattsburgh-NY
Poughkeepsie-NY
Reading-PA
Rochester-NY
SaranacLake-NY
Schenectady-NY
SiteR-PA
StateCollege-PA
Syracuse-NY
Teterboro-N
Trenton-NJ
Utica-NY
Washington/Dulles-DC
Washington/Natl-DC
Watertown-NY
Westhampton-NY
WhitePlains-NY
Wilkes-Barre-PA
Williamsport-PA
WillowGrove-PA
Wilmington-DE
WindsorLoc-CT


Slowest
Least Used
Most Used (thickness), city names alphabetized
#.# = hours
Fastest
Adelaide
Albury
AliceSprings
Ballarat
Bathurst
Bega
Brisbane
BrokenHill
Broome
Bundaberg
Burnie
Cairns
Canberra
Carnarvon
CoffsHarbour
CooberPedy
Coolangatta
Cowes
Daintree
Dampier
Darwin
Devonport
Dubbo
Euroa
Fremantle
Geelong
Geraldton
Gladstone
HaymanIsland
Hobart
Kalgoorlie
Katherine
Launceston
Mackay
Maroochydore
Melbourne
Mildura
Moe
MountIsa
Newcastle
Oodnadatta
Orbost
Perth
Queenstown
Rockhampton
Sydney
Townsville
Wangaratta
Warrnambool
Wodonga
Wollongong


Slowest
Least Used
Most Used (thickness), city names alphabetized
#.# = hours
Fastest
Albany-NY
Allentown-PA
Altoona-PA
Ambrose-NY
AtlanticCity-NJ
Barnegat-NJ
BeaverFalls-PA
Binghamton-NY
Blairsville-PA
Bradford-PA
Bridgeport-CT
Buffalo-NY
Danbury-CT
Dansville-NY
Dover-DE
Dubois-PA
Elmira-NY
Erie-PA
Fairfield-NJ
Farmingdale-NY
FortDrum-NY
Franklin-PA
GlensFalls-NY
GriffissAFB-NY
Groton-CT
Harrisburg-PA
Hartford-CT
Islip-NY
Ithaca-NY
Jamestown-NY
Johnstown-PA
Lakehurst-NJ
Lancaster-PA
Latrobe-PA
Massena-NY
MoguireAFB-NJ
Middletown-PA
Millville-NJ
Monticello-NY
Morristown-NJ
Muir-PA
NewHaven-CT
NewLondon-CT
NewYork-NY
NewYorkJFK-NY
NewYorkLGA-NY
NewarkIntl-NJ
Newburgh-NY
NiagaraFalls-NY
NorthPhiladelphia-PA
Ogdensburg-NY
Oneonta-NY
Philadelphia-PA
Pittsburgh-PA
Plattsburgh-NY
Poughkeepsie-NY
Reading-PA
Rochester-NY
SaranacLake-NY
Schenectady-NY
SiteR-PA
StateCollege-PA
Syracuse-NY
Teterboro-NJ
Trenton-NJ
Utica-NY
WashingtonDulles-DC
WashingtonNatl-DC
Watertown-NY
Westhampton-NY
WhitePlains-NY
Wilkes-Barre-PA
Williamsport-PA
WillowGrove-PA
Wilmington-DE
WindsorLoc-CT


Slowest
Least Used
Most Used (thickness), city names alphabetized
#.# = hours
Fastest
Albany
Charleston
Annapolis
Atlanta
Augusta
Cheyenne
Columbia
Columbus
Austin
BatonRouge
Bismarck
Boise
Boston
CarsonCity
Harrisburg
Dover
Denver
DesMoines
Concord
Frankfort
Hartford
Helena
Indianapolis
Jackson
JeffersonCity
Lansing
Lincoln
LittleRock
Madison
Montgomery
Richmond
Raleigh
Pierre
Providence
Nashville
NewYork
OklahomaCity
Olympia
Phoenix
Montpelier
Sacramento
Salem
SaltLakeCity
SantaFe
Springfield
St.Paul
Tallahassee
Topeka
Trenton
Washington

Judging Results

Trick or Treat

Performance based on the map
(1 = shortest path found, .5 = path found, but not shortest, 0 = no path found)

1. **Northeast (Rating: 2/3)**
    a. *Latrobe to Bridgeport*: found path, but not shortest
    b. *StateCollege to Dover*: found path, but not shortest
    c. *StateCollege to New Haven*: probably shortest, unable to verify [AK]

2. **Australia (Rating: 3.5/4)**
    a. *Melbourne to BrokenHil*l: found path, but not shortest (but very close!)
    b. *Dubbo to Bathurst*: shortest
    c. *Perth to Launceston*: shortest
    d. *Sydney to Townsville*: shortest

3. **US Capitals (Rating: 4/4)**
    a. *Columbus to OklahomaCity*: shortest
    b. *Indianapolis to JeffersonCity*: shortest
    c. *SaltLakeCity to Dover*: shortest
    d. *Baton Rouge to Montgom*ery: shortest

4. **Subways (Rating: 2/3)**
    a. *GCS to UnionSq*: shortest
    b. *155Amst to 103West*: not found (but link was there, though not clearly visible due to the low print resolution)
    c. *181Wash to 18West*: shortest

Average Rating: 82.1% Shortest paths found!

Alphabet Soup

As we had expected from our Final Analysis, the number and elongation of links caused the map to appear more cluttered than desired.

The results were generated using the player itself to output all shortest paths to a file and then used the grep tool to find the specific path based on source and destination.

Conclusion

We found that in general judges did not care about geographical accuracy, and the only thing that actually determined whether the shortest path was found was the clarity of the links and de-cluttered labels. In conclusion we feel that our maps displayed significant amount of information but for this project there is no perfect way of displaying every shortest path.