Project 2: How Far Away?

Group 5
Hari Kurup (hgk2101@columbia.edu)
Chris Murphy (cdm6@columbia.edu)
Madhuri Shinde (mss2103@cs.columbia.edu)

October 20, 2004

Introduction

This project was based on a simple problem: conventional maps tell you how far away two places are by *distance*, but not by *time*. If there are two ways to get from point A to point B, you might be able to tell which one is *shorter*, but not which one is *faster*. The challenge, then, is how to come up with a map that effectively shows the shortest time to travel between two points, given a data set of cities and links between them. The map would have to be generated by a software program given only the data set, and would have to fit on one piece of letter-size paper.

This document starts off by explaining our initial thoughts about this problem, based on our discussions as a group and in class. It then lists some of the approaches and strategies that we considered and prototyped, before discussing the implementation that we ultimately settled on. Finally, this report summarizes our results in the various user evaluation tests that were performed, and finishes with some closing thoughts and ideas on how we could have improved.

Background and Initial Findings

Based on our initial discussions as a group and in class, we felt the following were the most important ideas to use in this project:

- Originally we thought that this project was an exercise in finding the shortest path between two points, and we discussed search algorithms like A*. And many of the discussions in class focused on techniques that were *possible* but perhaps not *useful.* We soon realized that this is more of a user interface and usability problem: how do you effectively display the map when there is so much information? As such, we started approaching this by using usability techniques, focusing mostly on how well a user would understand the map.

- We also realized, however, that it is useful to consider this a “search” problem, because there can be great gains from removing unnecessary links (i.e. links that do not appear on the shortest paths between any two points). Though some other groups pointed out that the gain from doing so is minimal, we felt it was indeed worthwhile and that it could actually make the map easier to read.

- One of the most commonly discussed ideas in the class was to distort the map into "clusters" or "hubs". And while we did end up exploring this possibility in our first few versions of the code, we quickly agreed that it would be better to minimize geographical distortion. We were especially concerned that too much distortion would confuse users who already familiar with the geographic layout of the area, and we saw little gains in grouping things into clusters or hubs. Though some potential scenarios were discussed (e.g., a map showing the Paris subway system and the rest of the world), we assumed that the maps we would draw would all be on an equal scale (i.e., not composed of a small area combined with a big one).

- Many groups pointed out that it is rather impossible to always effectively show the shortest path between two points in such a restricted setting. The best that could be hoped for would be to minimize calculations the user must perform. We knew that the only way to accomplish this would be to make it clear to the user which links were fast and which were slow. This is discussed in more detail in the "Displaying time" section below.

- Some groups favored the idea of text representation of the data, as opposed to drawing an actual map. Though we recognize that this can be a useful way of demonstrating the shortest path between two points, we felt that the restriction of limiting the "map" (in whatever format) to one page made this impossible. We also thought that this approach would be useful for, say, engineers and computer scientists but might not be useful for other people.

Strategies and Concepts

This section describes some of the concepts and ideas that we experimented with and discussed past the initial stages. Though many of them were ultimately thrown out, we feel that a future project may want to explore some of these further.

Basic map (no distortion)

The very first map that we created simply drew the cities in their exact positions, maintaining complete geographical accuracy. This was a very simple place to start, and the map would appeal to people who were already familiar with the geography of the area. Our group felt strongly that the map should not be distorted to the point where the user could not easily find cities based on previous knowledge.

Unfortunately, the major problem with this map is that many cities might be too close to each other, or that the links would overlap and be difficult to read. Moreover, it would not be an effective use of all the space on the map. Therefore, we quickly learned that the main challenge in this project is how to separate the cities effectively, so that you have a balance of readability and geographical accuracy.

Spacing (making minor adjustments)

The first idea we had in an effort to reduce the amount of clutter on the map was to perform minor adjustments in how they are spaced out. We wanted to maintain geographical accuracy as much as possible, but there could still be occasions when cities were too close to each other for their labels to be readable. In those cases, we would try to move the cities away from each other, but not by too much.

Before a city would be drawn on the map, we looked at all the other cities that had already been drawn. If the new city were within 50 pixels of another city in either direction, then it was considered to be too close. We would then determine which axis the cities were to close on, and move the new city away by 20 pixels. Of course, that might not be enough, or that might put it too close to another city, so we would reduce the number of pixels to move the city by and try again. After five moves, we would accept that there was little we could do to reduce the clutter, and would stop.

The “Scaling” approach (described below) soon replaced this method, as we realized that the cities were generally still too close together and that the slight adjustments typically made little difference to the overall map. We abandoned this idea once and for all when we implemented the “Abbreviations” concept (see below), which gave us an easier solution without having the drawbacks of distortion.

Areas for future development

Though sometimes this produced visibly noticeable effects on some of our maps, it had very slight effects on others. One problem was that sometimes it would lead to large geographical distortion (thus undoing the work of our “Scaling” algorithm). A future implementation would probably need a smarter mechanism for separating cities that are too close. It could do this by finding unused space on the map, or by adjusting all other points in order to space out a small group.

Grid layout

On the opposite end of the spectrum from the minor adjustments approach, we also considered introducing massive geographical distortion in the hopes of making the best use of empty space and making the map as readable as possible. One idea was to put all of the cities into an *m* by *n* grid, evenly spaced out around the map. Though this could lead to very strange results (like Boston appearing to be closer to Washington, D.C., than to New York City), there could be great benefit from the fact that labels would not overwrite each other and that links would not overlap.

We immediately discovered that a grid with cities on right angles to each other would not be useful because links would appear to go through cities, even when they actually did not. This could happen on diagonals, too. To compensate for this, we added some slight variation in how the cities were lined up in the grid, so that cities close to each other would not have the exact same X or Y coordinates.

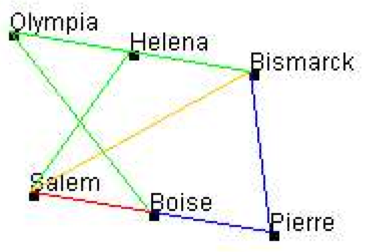


Figure 1. The cities are displayed using a grid layout.

The problem inherent to this approach, though, comes not only from the distortion of geography, but also from the confusion that could be caused by links having the same slope and thus appearing to be the same line. Even slight variances on the X and Y axes did not have too much effect because lines heading in the same direction were too easily confused. We eventually gave up on this idea, as the benefits were outweighed by the problems caused.

Areas for future development

It is likely very possible that some algorithm exists (e.g., a solution to the N-queens problem) for ensuring that no two points are on the same X or Y axes or are on any diagonal, but anything short of that is very likely to run into problems with overlapping links. A future attempt to develop this would need to take that into consideration, possibly in conjunction with the "Arcs" approach described below.

Insets

The restriction of having the map fit on one piece of paper introduced many limitations to what we could do, but one thing that was described at length in class was to expand certain areas by creating map "insets". That is, it might be possible to remove all the labels from a cluttered part of the map, and then redraw that section in an enlarged scale on another part of the page.

One challenge was in finding free space on the map so that the inset could be drawn, keeping in mind that the space used for the inset would have to be bigger than the region in the original map. We decided that we would get around this by always having just one inset, even though that necessarily meant shrinking the rest of the map.

The next obstacle came from finding the most densely populated area. We discovered some algorithms for calculating this in terms of *cities*, but city density was not usually the biggest problem. The biggest problem usually came from the *labels* and the *links*. We found it to be quite challenging to determine which area of the map was the one that was hardest to read, keeping in mind that city density does not always imply clutter.

We decided not to develop a prototype of this solution, partly because of the problems listed above, but also because using an inset necessarily meant shrinking the rest of the map, and we did not want to sacrifice the rest of the map just for one small area.

Areas for future development

We still feel that this is an idea that has a lot of potential, and another group may be able to successfully implement it beyond a proof of concept. As noted, the group would have to overcome two main problems: how to find free space on the map, and how to find the most cluttered area. For the latter some implementation of a clustering algorithm like *k-means* could be used. Once a cluster is found, the links between the cities in the cluster may be removed, and that region shrunk. This cluster region can them be expanded in the inset. We think that this idea could be used in conjunction with another (like the grid layout) that might be used to free up a certain spot where the inset could be placed.

Trunks

As we continued to look for ideas on how to reduce clutter around certain areas, and in particular around hubs (cities with many links), one idea was to introduce "trunks" that would contain many "branches". That is, rather than have all the links coming in to one city, some (or all) of the links could converge on a point away from the city in a more open space on the map, and there would be one thick line (the "trunk") connecting that point to the city.

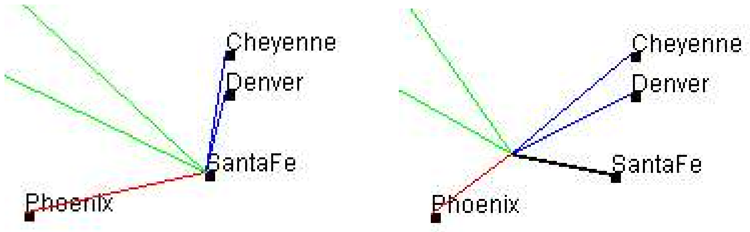


Figure 2. The picture on the left is without the trunk. In the picture on the right, the trunk is the black line connected to Santa Fe. It removes the clutter from around the Santa Fe area and makes the links from Cheyenne and Denver easier to see.

The main advantage of this would be that there would be fewer lines converging on one point (the city), and we could possibly have multiple trunks to eliminate the problem of just moving that convergence point to another place on the map. However, we decided not to pursue this idea because it might be difficult to choose a good spot to place the trunk (we would need to find a somewhat open spot on the map that was near the city) and it would be somewhat confusing for the user if the trunk were located too far from the city. There were also conceivable problems with our coloring scheme, which is described in the next section.

Areas for future development
We still think this is a worthwhile idea and one that could be used effectively (we think that some other groups considered similar ideas), but, as explained previously, the main hurdle to overcome would be the placement of the trunk and how to ensure that users would not be confused by many links converging on a place that is not their intended destination.

Arcs

It seemed that every group was using straight lines to indicate links between cities, but the TA suggested that it might be possible to use curved lines (or "arcs") instead. We thought that this might be useful in situations where links were overwriting each other, especially if the links were somehow labeled and the overlapping labels became too difficult to read. We also considered using arcs to show that some links were longer than others (in terms of time), which might provide a visual cue to the user.

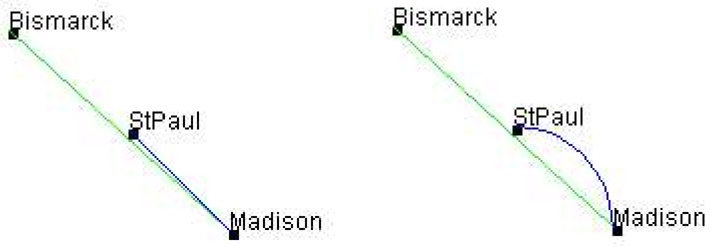


Figure 3. The image on the left shows how the link from St. Paul is obscured by the link from Bismarck. The image on the right uses an arc from St. Paul to make it clearer that there is a link there.

After a short trial, though, we decided not to pursue this idea any further, primarily because of the complexity of drawing an arc between two points using the Java 2D API. We also realized that the arc may end up passing through another city if not carefully drawn, and this could confuse the user. There seemed to be little reduction in overall clutter as well.

Areas for future development
One area in which arcs could be used in a future implementation is when three or more cities have the same X or Y coordinate. The links between them would appear to be straight lines and they would overlap each other, so arcs could be used to "go around" the intermediate cities.

3-D

This was an idea that no group had mentioned in class, and we admittedly were sitting on it until we could develop it a bit more. The basic idea was to try to render a 3-D map instead of a flat 2-D map. Whether or not anyone else actually thought of it, we don't know, but we thought it might be possible to represent relative distances or times by using the Z axis (while still maintaining geographical accuracy on the X and Y coordinates). Some possible limitations here include users' confusion as to the meaning of the different "height" of the cities, the fact that some cities might get hidden behind others, and difficulties in getting Java to easily render 3-D images (though we did not seriously investigate this).

Areas for future development
This idea might be used to reduce clutter by making some labels "higher" than others, or to somehow represent slow links by making them appear to be longer in 3-D. However, much consideration would need to be taken into effectively showing a 3-D map in a 2-D format.

Implementation

This section explains our algorithms for implementing the major features of the map-drawing program that we submitted.

Removing unnecessary links
As per the project specification, the dataset could contain a minimum of 100 links. Drawing all of these links on the map causes cluttering. Moreover the aim here is to provide a map with the fastest routes between cities. So, we decided to remove links that do not fall on the fastest route. This represents a graph theory problem that is analogous to the shortest path problem. We decided to use Djikstra's shortest path algorithm for our purpose. Normally, this algorithm works on the assumption that the graph is a directed graph and cycles are not present. In our case, this is not a reasonable assumption since there may be more than one route between a pair of cities (say, a train route and an air route), and according to specifications, each link is bi-directional. So we treat the graph as bi-direction with cycles and adapted the algorithm to it.

We start the algorithm by converting the dataset into a time-weighted graph. While creating the adjacency matrix, if we encounter more than one direct link between the same pair of cities, only the link with the least time-weight is maintained. This gives us the shortest (fastest in this case) link between those pair of cities, but we still need to use Djikstra's algorithm to get the fastest route between all pairs of points on the map.

Consider a weighted graph G = (V,E), Djikstra's algorithm works by maintaing a set of vertices S whose final shortest –path weights from the source s have already been determined. The algorithm repeatedly selects the vertex u E V –S with the minimum shortest-path estimate, adds u to S and relaxes all edges leaving u [CLSR, 2nd edition]. Because this algorithm always chooses the "lightest" or "closest" vertex in V – S to add to set S, it is considered a greedy algorithm [CLSR, 2nd edition].

The number of unnecessary links removed by the algorithm clearly depends on the dataset. For the northeast.game dataset which has a total of 152 links, this implementation of shortest path removes 27 unnecessary links, whereas for the uscapitals.game dataset it removes only 7 unnecessary links..

Scaling

In order to make the map fit the image for any set of coordinates, it was necessary to scale the range of coordinates to the dimensions of the screen, and move the coordinates into place. In addition, we hoped to avoid clutter by spreading out the cities a bit and trying to ensure that no two cities had the same X or Y coordinates. This idea was one that we considered in the first few days of the project (see "Grid Layout" above), but the actual implementation was influenced by a discussion with members of Group 6.

The concept that we used was the idea that the actual X and Y coordinates don't really mean anything; what is important is the relative placement of cities on the map. We sorted the cities into two lists: one by the X coordinate, and one by the Y coordinate. If two cities had the same coordinate, the value would be altered slightly. Then we reassigned coordinates: the first city would be 0, the next would be 1, etc. After doing this for both axes, we had a data set in which there would be little overlap and equal spacing.

Though this led to slight geographical distortion, we felt that the extra spacing was a worthwhile tradeoff, and would point out that cities are still geographically accurate with regard to each axis (e.g., Seattle would still be west of New York and Boston would still be north of Miami).

In order to determine the X and Y coordinates that would be used on the map, we then calculated an expansion factor (how to spread them out) and offset (how to provide a buffer in the map) as follows:

xFactor = xScreen/(maxX - minX);
*xOffset = xScreen - (xFactor * maxX) + 10;*

where xScreen is the width of the screen, minX is the minimum X coordinate (which would be 0 after repositioning), and maxX is the maximum X coordinate (which would be one less than the number of cities). The extra 10 at the end is so that there would be a 10-pixel buffer around the map. We then recalculated the city's coordinates like so:

*city.x = (xFactor * city.x) + xOffset;*
This method assured that the cities would be spread out across the map (to use as much of it as possible) and that they would be positioned in the right place.

We also need to rescale for calculations that might put our labels off-screen. This especially happens on the far-right (east) side of the map. In those cases, we try to move the label to be to the left of the city's location, instead of in its default spot to the right.

Abbreviations
In our final version of the map generator, we abandoned the "Spacing" idea (described above) and decided that the best way to reduce clutter would be to shorten the city labels. It occurred to us that the problem with a crowded map did not come from the links or even the location of the cities themselves, but from the labels used to demarcate the cities. These labels would often overwrite each other and though we initially came up with ideas on how to move those labels (based on the ideas expressed by Group 8), we thought that removing them altogether would be better.

We decided to use abbreviations of the cities' names at first, but then decided that it would be better to just replace those names with some code and then have a table aside the map that explained which code represented which city. If any city is within 40 pixels of another city, then its city name is added to a list of cities to be replaced. When the cities are drawn, then the city is given a new "name", which starts with the letter "C" (for "city") and is followed by a two-digit number. Additionally, any city whose name would go over the right-hand side of the map is added to the list.

After all the cities have been drawn on the map, a legend is created on the right hand side, mapping these new city names to the original name. The cities are ordered alphabetically so that the user can easily find the city if he knows the abbreviation, and can easily find the abbreviation if he is looking for a particular city. Currently we limit the number of abbreviated city names to 37, as that is all that would fit on an 800x600 image.

Though this may cause some confusion for users, we feel that the reduction in overall clutter in the map (and in particularly crowded areas) is a very worthwhile tradeoff. We also preferred this idea to the "Inset" concept (described in the previous section) because it would work on a wider scale by affecting more cities in more crowded areas, rather than just one crowded area.

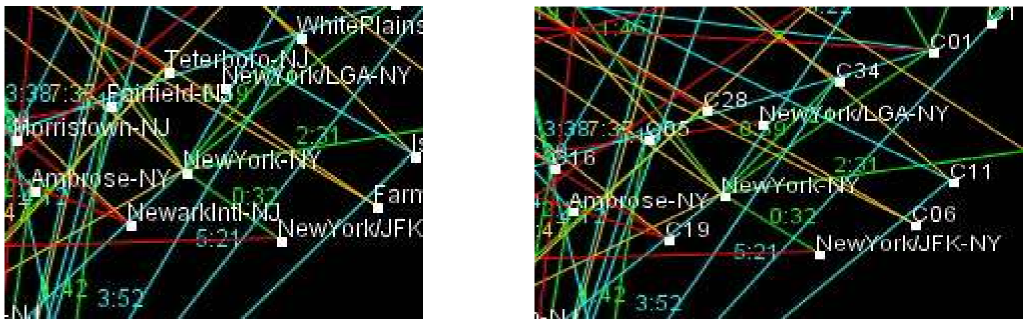


figure 4. On the left an area of the northeast map without abbreviation. On the right, the same region, with abbreviations.

Coloring links

One of the key features of our map is that it colors the links differently in an attempt to give the user a visual cue that some links take less time than others. We originally were coloring based on the *speed* of the link, but realized that the user would be more interested in the *time*, so we changed our map accordingly.

In order to calculate the color of each link, we calculate the “timeStep” variable. We find the minimum and maximum speeds of the links, and then divide the difference by four to get the “timeStep”. This roughly separates the links into four time categories (though the separation is obviously not going to be uniform). The value of “timeStep” obviously depends on the speeds used on the map, but for our US Capitals map, it was about two hours.

Links are colored as follows:

COLOR	**TIME**
green	shortest (0 – timeStep)
cyan	short (timeStep – 2*timeStep)
orange	long (2*timeStep – 3*timeStep)
red	longest (3*timeStep – 4*timeStep)

We used the color green to give the user a feeling of “going”, whereas red was meant to give the user a feeling of “stopping” (or moving slowly). We considered using three or five colors, but felt that four was a better number because it gave the user few colors to have to remember, but still could show a good range of times.

The time of each link is shown on the map as a label in the same color as the link itself. However, times are not shown for red links because the time is usually so long that the user will generally want to avoid that link if possible. We experimented with removing the labels for other colors, but found many situations in which a user might miscalculate the shortest path. The time is shown halfway between the two cities it connects. In some cases (when the link goes from southwest to northeast) we move the time to the left of the line so that it can be displayed more easily than in its default position, which is to the right.

The first iterations of our map used a white background. We felt that a black background would be better because the links would be easier to see, the cities would stand out more, and the user could more easily differentiate labels when they are cluttered. This also allowed us to use the same color for the time labels as for the links themselves, which did not appear as clearly on a white background. We also experimented with a gray background, but the colors were not bright enough to contrast as well as they did against black.

Tournament Maps

The maps generated by our project for the four datasets, 3 known and 1 unknown that were part of the tournament are given below:

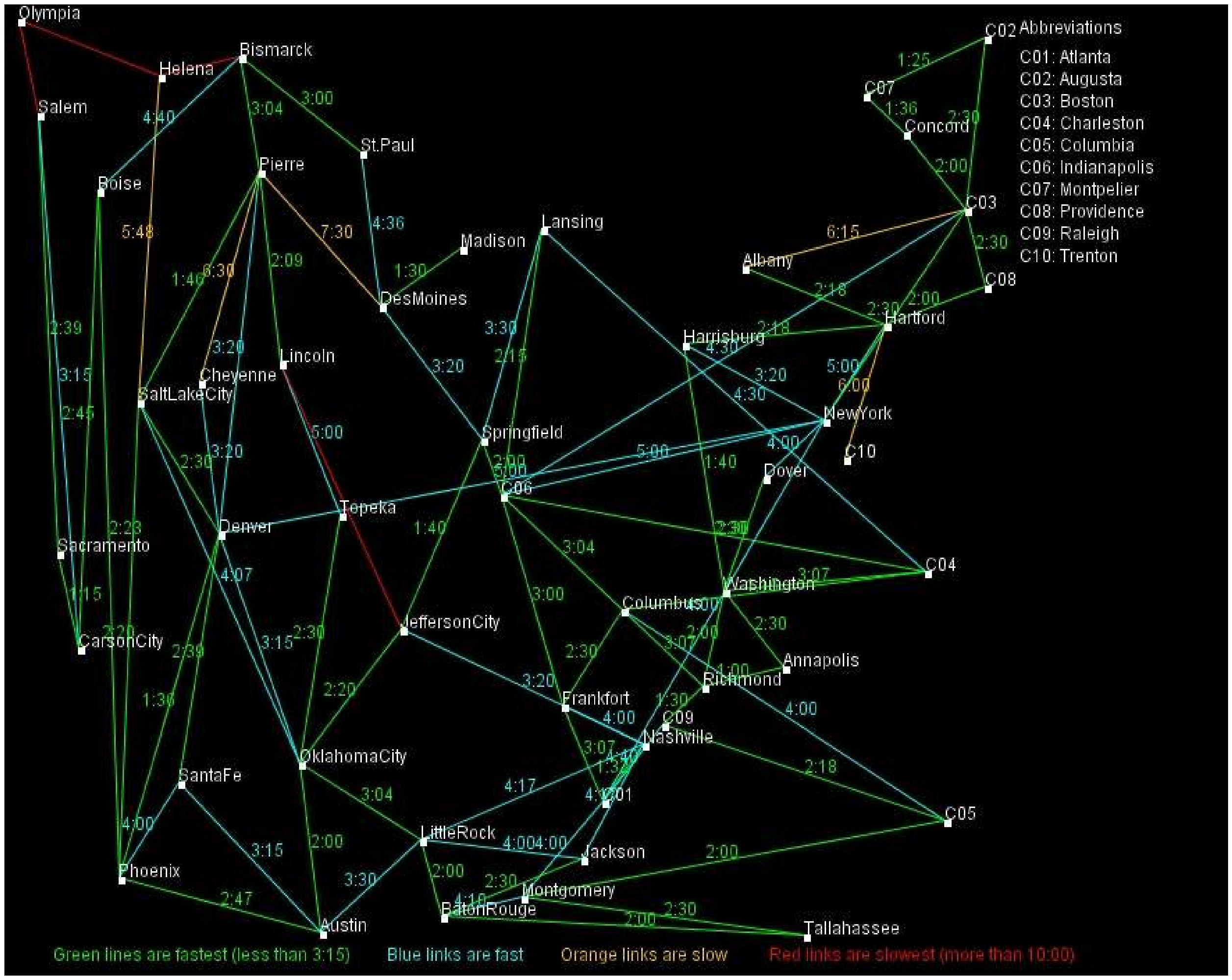
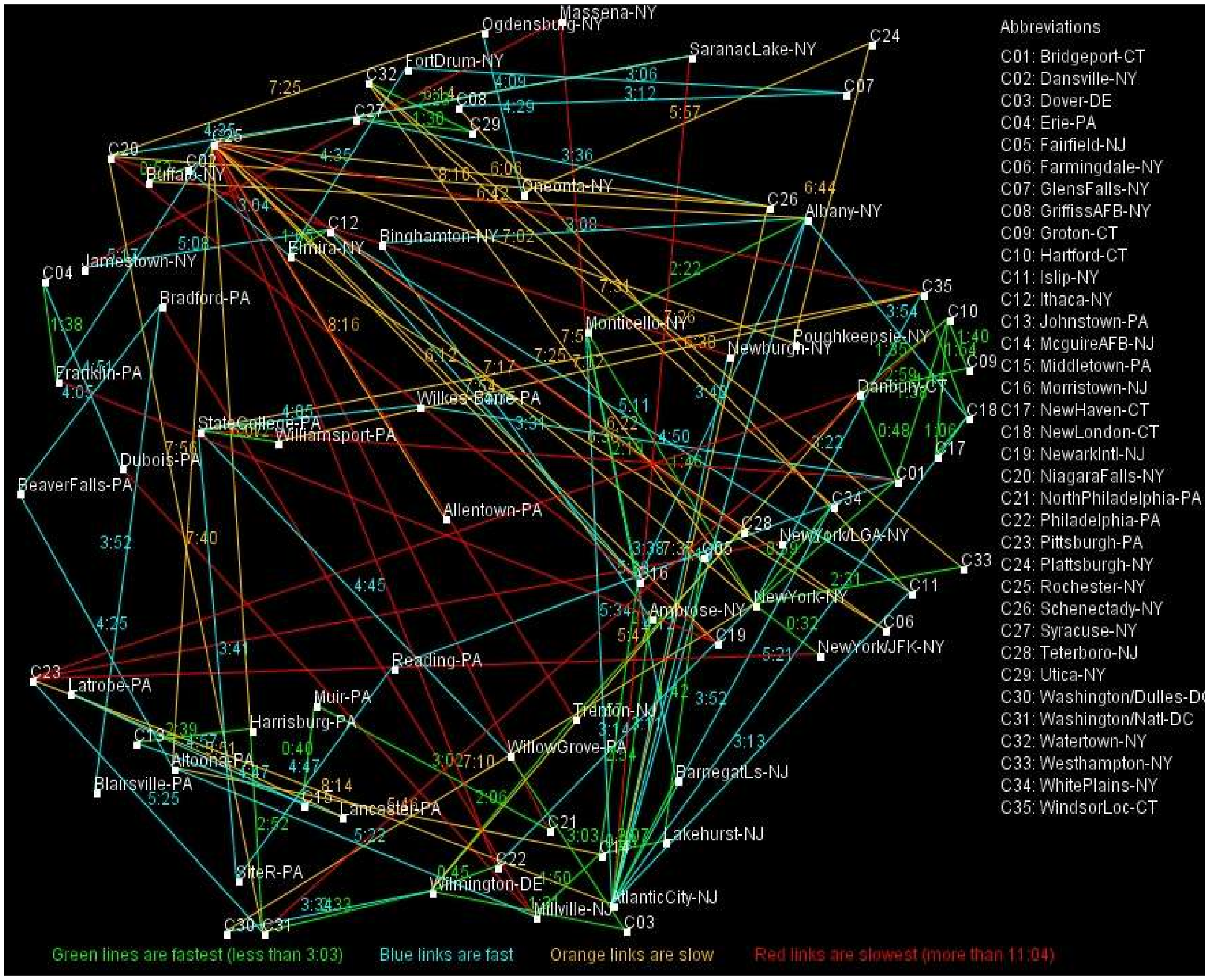
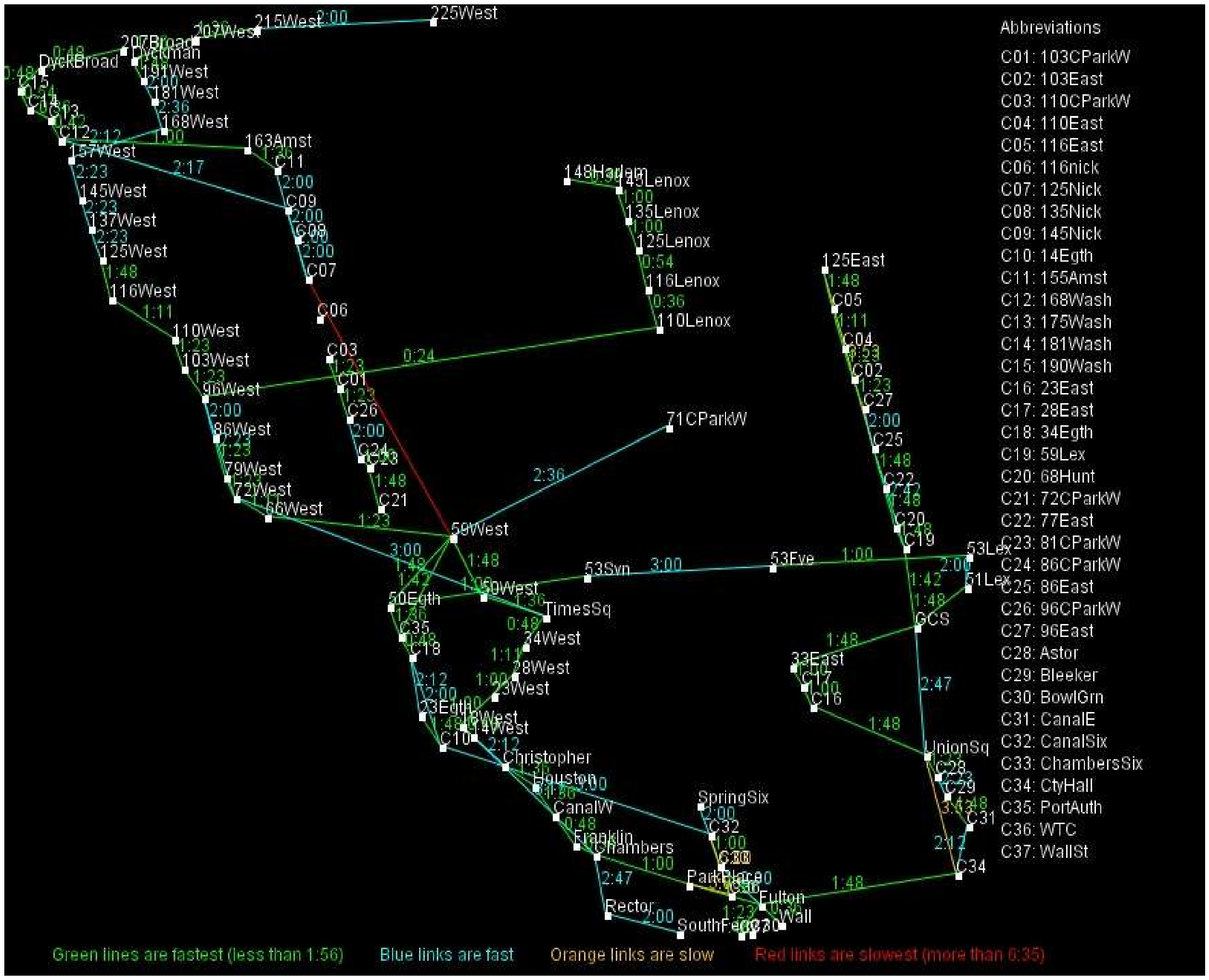


Figure 5. A map of capital cities of the United States.


Green lines are fastest (less than 3:03)
Blue links are fast
Orange links are slow
Red links are slowest (more than 11:04)

Abbreviations
C01: Bridgeport-CT
C02: Dansville-NY
C03: Dover-DE
C04: Erie-PA
C05: Fairfield-NJ
C06: Farmingdale-NY
C07: GlensFalls-NY
C08: GriffissAFB-NY
C09: Groton-CT
C10: Hartford-CT
C11: Islip-NY
C12: Ithaca-NY
C13: Johnstown-PA
C14: McguireAFB-NJ
C15: Middletown-PA
C16: Morristown-NJ
C17: NewHaven-CT
C18: NewLondon-CT
C19: NewarkIntl-NJ
C20: NiagaraFalls-NY
C21: NorthPhiladelphia-PA
C22: Philadelphia-PA
C23: Pittsburgh-PA
C24: Plattsburgh-NY
C25: Rochester-NY
C26: Schenectady-NY
C27: Syracuse-NY
C28: Teterboro-NJ
C29: Utica-NY
C30: Washington/Dulles-DC
C31: Washington/Natl-DC
C32: Watertown-NY
C33: Westhampton-NY
C34: WhitePlains-NY
C35: WindsorLoc-CT

Figure 6. A map of some major cities and regional airports of the northeast region of the US



Abbreviations

C01:	103CParkW
C02:	103East
C03:	110CParkW
C04:	110East
C05:	116East
C06:	116nick
C07:	125Nick
C08:	135Nick
C09:	145Nick
C10:	14Egth
C11:	155Amst
C12:	168Wash
C13:	175Wash
C14:	181Wash
C15:	190Wash
C16:	23East
C17:	28East
C18:	34Egth
C19:	59Lex
C20:	68Hunt
C21:	72CParkW
C22:	77East
C23:	81CParkW
C24:	86CParkW
C25:	86East
C26:	96CParkW
C27:	96East
C28:	Astor
C29:	Bleeker
C30:	BowlGrn
C31:	CanalE
C32:	CanalSix
C33:	ChambersSix
C34:	CtyHall
C35:	PortAuth
C36:	WTC
C37:	WallSt

Figure 7. A map of some stops on the New York City subway.

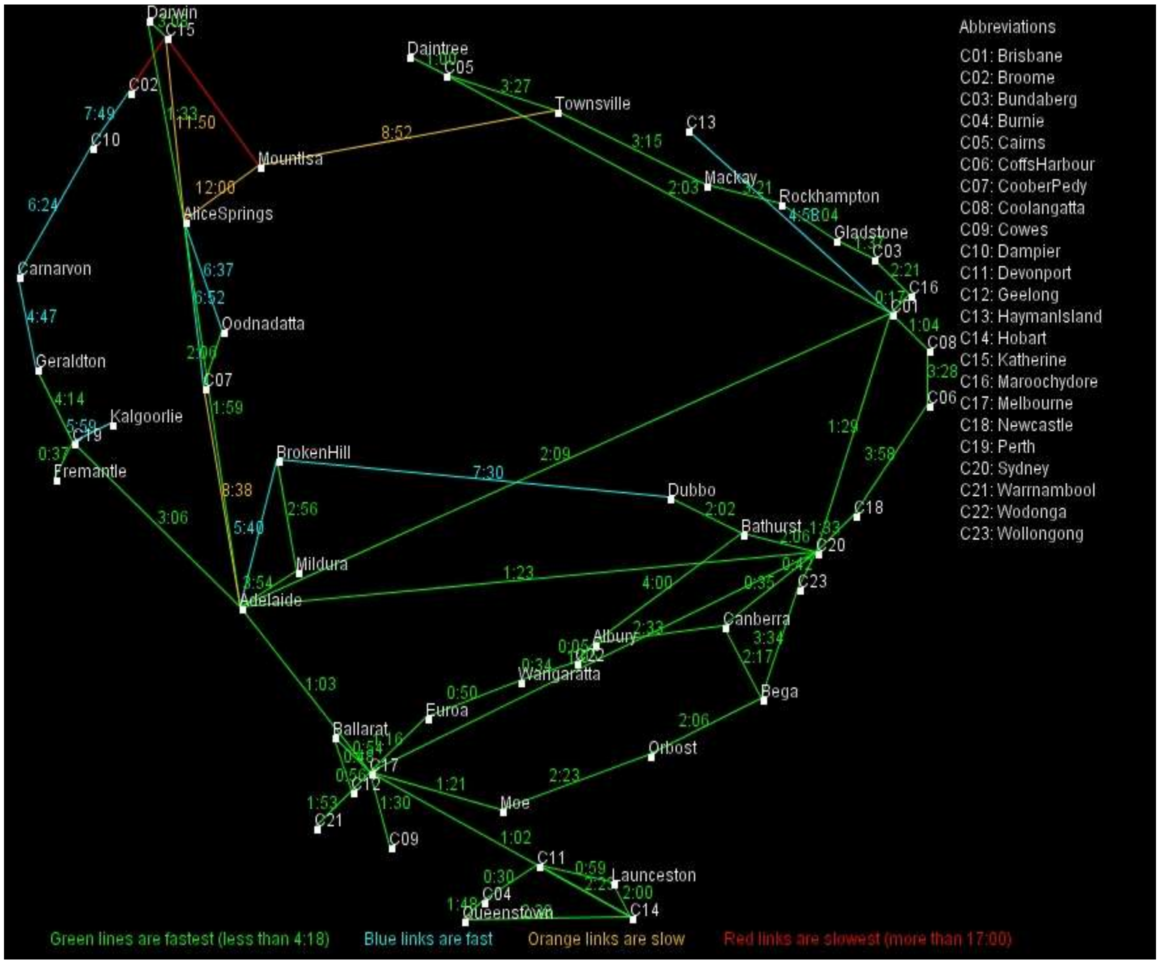


Figure 8. A map of Australia.

User Evaluation Results

Out performance in the user evaluations were good. We managed to get some good reviews, and in most cases our map was legible enough for the judges to find the routes. Here are a few comments about our maps from the judges:

- Color legend helpful.
- ("the legend was good")
- Direct link (4:10 time noted)

Not all reviews were positive. There were some that made us realize that more work needs to be done in certain areas. For example, on the map of the northeast, one of the judges could not find a path between New York and Reading because he found map to be cluttered. This tells us that there is scope for improvement in the area of congestion reduction.

The US capitals map and the secret map (Australia) were our best maps in the tournament. Almost every judge found it easy to use and find the fastest route. The subway map comes in next with only one mishap. The map of the northeast, though cleared of lot of clutter still had some.

Conclusion and Lessons Learned

Areas for improvement

It occurred to us early on – and it is repeatedly mentioned throughout this document – that reducing clutter is the key to implementing a good map. Regardless of technical approach, the problems of overlapping links and link times, and cities too close together, are ones that must be dealt with in order to produce a map that is ultimately usable. While maintaining geographical accuracy is *desirable*, we learned that making the map readable is *essential* from the user perspective. Therefore, we feel that more emphasis should be put on making the map easy to read, even if it means distorting the geography.

Accomplishments

Despite the fact that we could have made the map easier to read, we still feel that this project was successful for the following reasons:

- We achieved some degree of spacing while still maintaining geographical accuracy. By using the "Scaling" approach, **we managed to find a balance** between making the map easy to read and ensuring that the cities were positioned in places familiar to the user.
- We took a **user-centric approach** instead of a technology-centric one, by trying to think about "what should be done" instead of "what could be done". Whereas it seemed to us that many other groups were considering the technological ease of the solutions, we realized that the end-user of the map was more important, and we took that in mind at each step.
- We explored and documented **a number of other concepts** that could be developed by other groups in the future. Though many of these were not implemented in this project, we feel that many of them have a great amount of potential for the future.

Acknowledgements

- The idea for placing each city on a grid with each having a unique X and Y comes from Group 6.
- The idea for moving the city label based on its proximity to another city label came from the concept expressed in class by Group 8.
- Implementation of Djikstra's algorithm, adapted from Data Structures in Java, 2nd edition, Robert Lafore.
- Some of the ideas for heuristics for making the map more user-friendly came from Jakob Nielsen's "Ten Usability Heuristics" (http://www.useit.com/papers/heuristic/heuristic_list.html)
- Working principle of Djikstra's algorithm taken from Introduction to Algorithms, CLSR, 2nd edition.