Kuncheng Feng, Taeyoung Park, and Johnson Liu

CSC366

Professor Schelgel

12/3/2021

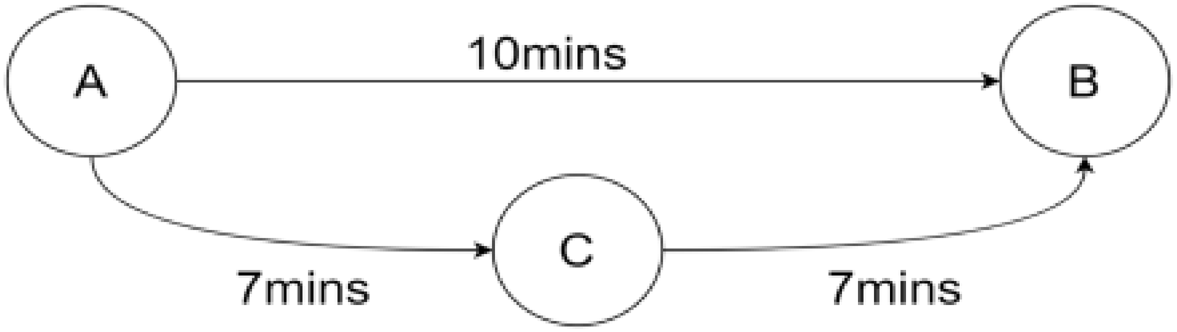
# Train Network Cognitive model

## Introduction

In our modern world, people travel all the time. Whether it's from class to class, building to building, or country to country, moving from one place to another is a norm. But changing locations isn’t a simple thing that happens in the blink of an eye. Before they set foot on their path, people consider many things that determine which path they should take. For our final project, we wanted to model the cognitive process that people go through in order to decide which is the best path to take. After some discussion, we wanted to model a world of train networks that includes stations and routes that connect them, and our commuter will have to decide which route is the best to take when traveling from one location to another.

## Background

Our model is basically a graph problem, unlike the graphs that we are familiar with in high school classes where points are plotted based on X and Y values to show relations. “A graph in this context is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines)” (Graph theory). And the problem that involves these graphs is finding out which edges connect two nodes with the lowest weight. Computer scientist Dijsktra’s “original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the ‘source’ node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree” (Dijkstra’s algorithm).



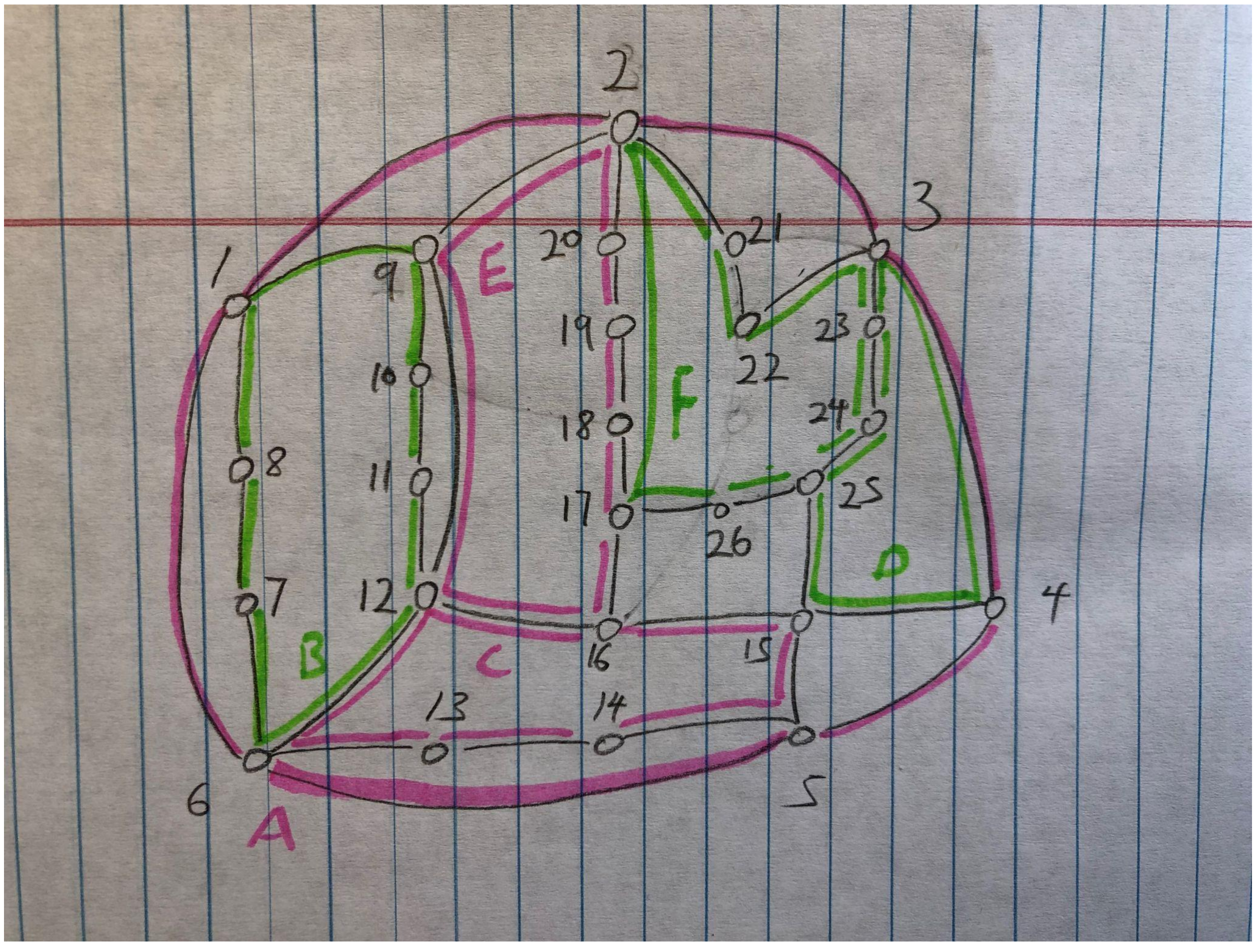
Facebook, YouTube, GPS, Yelp, or any other services that recommend their users similar content, have algorithms that solve the graph problem on a daily basis. In YouTube's case each video is considered a node, and the links to other videos are edges, while the user is watching an video, YouTube's algorithm will take many factors into account and recommend the users the links to other most similar videos, these links have their version of highest weight, or graph theory's version of shortest path. For Google Maps, they collect data from cell phone users, and then analyze "the total number of cars, and how fast they're going, on a road at any given time." (Stenovec). And then literally recommends the user the shortest path. And lastly for us, the train stations will be nodes, and the routes that connect them are edges, upon consulting with the commuter's knowledge about the network, a shortest path will be determined.

## Methods

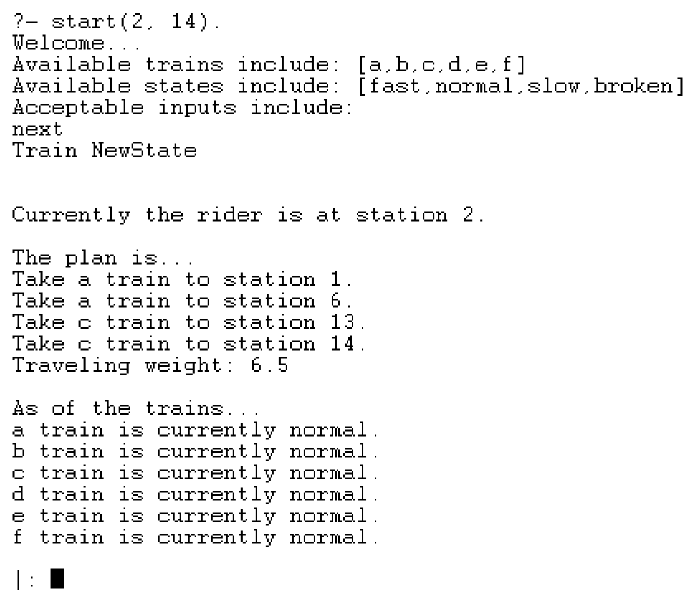
In our implementation, we modeled our world with stations, trains, connections, and an abstract rider. The rider holds the information of where he is currently at, where he needs to go, and a knowledge base about the world. The knowledge base can be organized in two categories: hard knowledge and soft knowledge. Hard knowledge base consists of stations, trains, and the connections that indicate which station is connected to which, by which train and by what weight. Soft knowledge base consists of the state of trains that will later be used to modify the weight of connections, and along the rider's journey he will have chances to revise his beliefs about the trains.

When the program starts, the rider first needs to know his starting station and his goal station, he then gathers up all the hard and soft knowledge with the assumption that all trains have normal status, and finally begins his journey. In between train rides, the rider will imaginary travel all the possible paths to reach his destination, whether he learns new information or not, he then weights those paths with the basic traveling weight from hard knowledge base and modifies it with the states of the trains from the soft knowledge base. The path with the lowest weight is the one that the rider will resume on. The new information he might learn revolves around how fast a train is going, or sometimes if a train is functioning or not. At some point the rider will choose a path with more in between stations over a path with less, because he believes that the local route is actually faster or that the train that runs the express route is broken. If he has no paths to go he will be stuck, else if he reaches his destination the program will end.

## Results

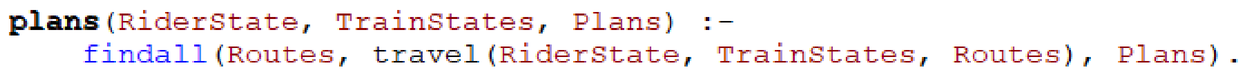


The above image is our final version of the train network, it consists of 26 stations (1 ~ 26), 6 trains (A ~ F), and many connections. Each connection's weight is embedded in the code, but the general rule is that a local station always has the weight of 1, and if it takes X weight to get from a station to another with local routes, then the express route will have the weight of X divided by 2. And just a side note each transfer of train costes 1 weight to make it more realistic.

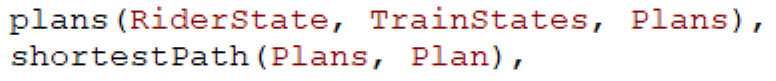


```
?- start(2, 14).
Welcome...
Available trains include: [a,b,c,d,e,f]
Available states include: [fast,normal,slow,broken]
Acceptable inputs include:
next
Train NewState


Currently the rider is at station 2.

The plan is...
Take a train to station 1.
Take a train to station 6.
Take c train to station 13.
Take c train to station 14.
Traveling weight: 6.5

As of the trains...
a train is currently normal.
b train is currently normal.
c train is currently normal.
d train is currently normal.
e train is currently normal.
f train is currently normal.

|: █
```

At the start of the program, the rider needs to know his starting location as well as his destination, and his entire knowledge base is printed out to the user to get a better sense of what is going on. In this demo the rider is trying to get from station 2 to station 14, he assumes everything is normal, then considers all the available options he has, and finally takes the best option out of it.

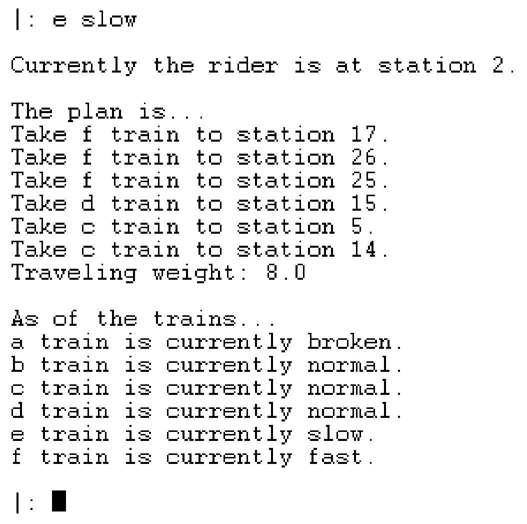


```
plans(RiderState, TrainStates, Plans) :-
    findall(Routes, travel(RiderState, TrainStates, Routes), Plans).
```



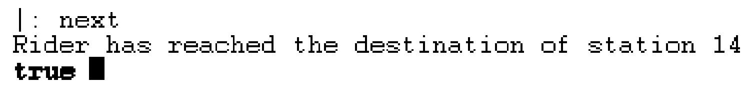
```
plans(RiderState, TrainStates, Plans),
shortestPath(Plans, Plan),
```

Before the rider actually makes a move, he has chances to learn information and uses it to revise his belief about the best route to his destination. In this demo we told him that the A train is broken, the E train is slow, and the F train is fast. The following image is the rider's belief of the best path after his knowledge base is revised.



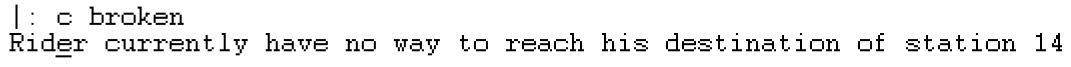
```
|: e slow

Currently the rider is at station 2.

The plan is...
Take f train to station 17.
Take f train to station 26.
Take f train to station 25.
Take d train to station 15.
Take c train to station 5.
Take c train to station 14.
Traveling weight: 8.0

As of the trains...
a train is currently broken.
b train is currently normal.
c train is currently normal.
d train is currently normal.
e train is currently slow.
f train is currently fast.

|:
```

To make the rider travel to the next station, enter “next” into the program. In this demo after entering many “next” commands the rider was able to reach his destination.



```
|: next
Rider has reached the destination of station 14
true
```

And of course the most fun part is that sometimes the rider can be stuck at a station when all the available trains of that station are broken.



```
|: c broken
Rider currently have no way to reach his destination of station 14
```

## Discussion

The biggest problems we encountered while completing this project besides the lack of flavors were exponential growth of complexity and run times as well as our lack of prolog skills. With one addition of connection to our model the brute force way of finding the shortest path just received multitudes of calculations. Our first model only consisted of 16 stations and the brute force method was able to find the shortest path in a blink of an eye, but just adding in 10 more to our current size the program is already noticeably a lot slower. An implementation of a different path finding algorithm can significantly improve our program. As for our lack of prolog skills, the majority of our time spent on this project is debugging something that’s not working, or restructuring our program to fit prolog’s binding and rebinding style of flow of information. And lastly, since we spend so much time on just trying to get this program to work properly, we did not have much time put into coming up with flavors of the world.

## Conclusion

Overall, we were able to successfully model the simplest form of cognitive process of choosing the shortest path. Which is to take what we already know, use the new things we learned to revise the weight of it, and then make decisions again after the weight of our knowledge base has been modified. With our capabilities as well as our available times we are proud of what we have developed.

## Works Cited

"Dijkstra's algorithm." Wikipedia, Wikimedia Foundation, November 02, 2021, https://en.wikipedia.org/wiki/Dijkstra's_algorithm.

"Graph theory." Wikipedia, Wikimedia Foundation, October 27, 2021, https://en.wikipedia.org/wiki/Graph_theory.

Stenovec, T. (2015, December 18). Google has gotten incredibly good at predicting traffic - here's how. Business Insider. Retrieved November 3, 2021, from https://www.businessinsider.com/how-google-maps-knows-about-traffic-2015-11.