Rough Draft

Introduction

Our project is based on commuters traveling from one station to another. The basic modeling world consists of trains, train stations, and train states. Each station is connected to an adjacent station. Commuters are given the option to decide the most suitable route and train to reach the target station from the starting station. We tell commuters the shortest route from the starting station to the destination station, and if the train is dirty or clean, they can decide whether to take or not the train. With our belief revision, we provide commuters with optimal options and make decisions accordingly.

Background

The problem we have selected for modeling a problem involving belief revision was can a rider travel from one station to another. The modeling world will consist of train stations, train routes, and a commuter. Each station will include at least one route, and the routes will connect at least two stations together. The problem we are trying to solve is what if we are the rider and we are trying to get from one station to the other. We will have to start at one station accompanied by a set of beliefs that help us to decide which routes and trains are best suited for traveling to the destination. And these beliefs can be revised when we gather more information along the way, such as if a train is too crowded then we might want to wait for the next train, or if we learn that a certain route is closed for whatever reason then we might

want to rethink what routes to take.

This problem is basically a graph problem, as each component of the world we are modeling can be simplified down to graph elements. Just to clarify, in this context the graph does not refer to the most common ones we see in business presentations or in data comparisons where a diagram with x axis, y axis, and plotted points are shown to better represent the relationship between quantitative variables. Instead we are referring to the graphs in mathematical graph theory, where "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). Our train stations can be represented by vertices, and our train routes can be represented by edges since they serve the same purpose. As of the rider's status all we need to represent is an hypothetical idea of where he is at. In graph theory there is also a distinction between directed and undirected graphs, since our abstract route is going to run two ways so our graph representation will be undirected. As of the rider's best route it can be represented by the shortest edge between vertices, and the edge in this sense will be the fastest route.

Fortunately for us graph theory has been studied way earlier than us and by people way smarter than us, what's even more fortunate is that the applications of it

have already been in use. Such as directional guide of GPSs, YouTube's recommended videos, FaceBook's friend recommendations, and many more. Now it may seem like just finding paths between train stations has nothing to do with YouTube's algorithms, but usprisinging they indeed follow the same fundamental concept that derived from Dijsktra's algorithm. 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). In GPSs case, they use the speed as the criteria to determine the shortest path, for YouTube each video is a node and some algorithm that determines its similarity with other videos gives the criteria for measuring the shortest path, in our case it's a lot simpler just being the distance between stations.

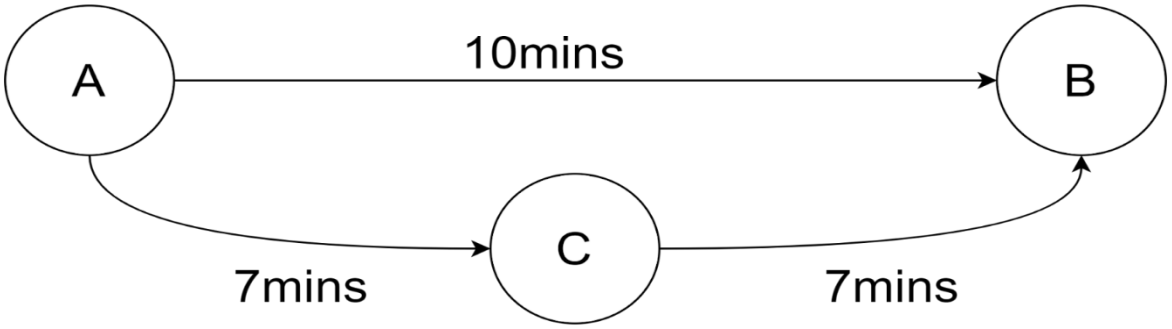
Believe revision in our situation is when the rider receives new information and uses it to determine if a route is still the best route to take. Fortunately GPSs already have this system implemented well, for example google collects data from cell phone users, and then "analyze the total number of cars, and how fast they're going, on a road at any given time." (Stenovec). From this the shortest path or the best route to take is influenced and subsequently recalculated, and updates the user the best path

which google believes the user should take. In our case when the rider learns something such as a crowded train or announcements of a closed route, he will have to rethink his believed best path with the new information he has.

While we were searching for research projects that are related to our research proposal we came across an interesting research article from MIT about belief propagation and networks with loops. The research paper discusses how, today, we already have proposed belief propagation rules to generate optimal beliefs for singly connected networks, and there have been a number of researchers who have already come up with good performing belief propagation rules to deal with networks with loops. However, the theoretical understanding of the belief propagation rules for networks with loops has yet been achieved. Therefore, the goal of the research project is to “lay a foundation for an understanding of belief propagation in networks with loops” (Weiss). In their project the researchers have experimented with networks with a single loop and networks with multiple loops. While working with networks with a single loop, they have discovered “an analytical relationship between the steady state beliefs in the loopy network and the true posterior probability” (Weiss). For networks with multiple loops, the research team has developed a concept called “balanced network" and illustrated simulation results by comparing belief

revision and update in networks with multiple loops (Weiss). We have selected this research paper because it relates to our project in the sense that our project also involves working with networks, specifically a train network. We plan on using the MTA train map as a model to construct our network and we assume that our network will involve loops. We hope that we can utilize this research paper as a reference to develop a train network involved in our belief revision project.

Since our problem is for subway users to find the shortest path. There is an interesting article about time-dependent stochastic shortest path(s) from Nottingham Trent University. They said that finding the shortest path in an optimal way requires multiple algorithms. For this they used a method called time-dependent stochastic shortest path(s), which is a combination of time-dependent path and stochastic path. The time-dependent path literally means finding the shortest path according to time. For example, let's say we are at location A and there are two ways to get to location B. If we go directly from location A to location B, it will take 10 minutes, and if we go from A to C through B, it will take 14 minutes. If so, the shortest time would be the first one.



Stochastic path is a probability variable that influences our decisions. For example road construction in the subway, congestion, absence of drivers, etc. If so we need to choose another path. Therefore, The time-dependent stochastic shortest path(s), which combines these two methods, is related to our Believe revision and we will refer to this research.

Method

When we first started the project, we were going to model our world over a real city transportation system, and we first decided that the NYC train network would be a good choice since it is a well-established system with the information being publicly available. But upon implementing it we realized that our initial plan was too ambitious. The system itself was huge, according to Wikipedia there were 472 stations and 36 lines. The connections are complicated as some stations have different names because they are actually a few stations connected together. Plus, sometimes the name duplicates but they are actually different stations, for example 4 lines stops at “125 street”, but these are 4 different stations at 4 different locations. Because of this we decided to tone down the scale of world representation and focus on getting the more important part of belief revision, then when we have the time,

we can make the system more complicated later. We came up with some conditions for how to represent our beliefs. As a basic concept, we created our own train network and there are 3 trains, each train circulates in a specific section. Each train has a specific condition when it stops at a station. This condition allows the user to decide whether to ride the train or not. There can be various conditions for this such as dirty, smelly, slow, etc. In addition, each condition should have a detailed description. The condition of dirty must have details such as there is a sandwich on the train or someone vomiting. The smelly condition has details such as a skunk is on the train, or someone is eating food. The slow condition has details such as the train has limited speed, or it changes local trains. For these reasons, the train causes a certain state, which allows users to decide whether to ride the train. The train status or conditions that we have come up with thus far are dirty, crowded, delayed, and suspended. Depending on the train status or condition, the commuter may need to re-route or change their travel plans. Each of the train conditions that we have organized would be caused by some sort of event or maybe even a sequence of events. For instance, the train would have the train status of “dirty” if the train carts have not been cleaned for a week and people have been leaving behind garbage on the train. Another possible event that would cause the train to be dirty is when it is snowing or raining since commuters would bring wet clothings, shoes, and umbrellas onto the train which would cause the train to be wet and dirty. Another train status that we have come up with is crowded. Some factors that would cause the train to be crowded would be that the train is running at rush hour, the train is the only available train, or an event is happening at a location that only the train makes a stop at. Next, for the train status to be in “delay” we came with a few possible situations. First, the train would be in delay if there is construction happening on a train track that the train is running on. Or if an accident has happened such as a person falling onto the train track it would also cause a train delay. In

addition, a flooding of a train station is another event that we plan to use to cause the train to be delayed. Finally, we plan to have a suspended train status where a train is no longer running so the commuter is forced to take another train in order for them to reach their destination. There are several events that we have developed that would make the train in our program suspended. One of the situations or reasons is that the train company is working on an improvement project for the train and its respective train stations, thus the train will no longer be available until the improvements are completed. Another reason that we could suspend the train is to say that the train is too old to function anymore

Discussion

This week we have implemented our mini train map model that we created last week into a prolog program. Our mini train map model consists of 3 trains (Trains A, B, and C) and 16 train stations (Stations 1 to 16). Train A makes stops at the following stations: stations 11, 4, 5, 6, 7 and 8. Train B makes stops at stations 1, 2, 3, 4, 11, 12, 13, 16 and 14, and train C makes stops at stations 7, 9, 10, 11, 12, 13, 14 and 15. Train A runs in a single-line train path and in both ways from stations 11 to 4 to 5 to 6 to 7 and finally to 8. Train B runs on a circular train path and in both ways from stations 1 to 2 to 3 to 4 to 5 to 11 to 12 to 13 to 16 to 14 and back to 1. Train C also operates on a circular train path and in both ways from stations 7 to 9 to 10 to 11 to 12 to 13 to 14 to 15 and back to 7. Some of the trains are purposely designed to make similar stops so that train transfers can be done. These information are made into facts in our prolog program. After we have established the facts in

our prolog program. We began to create a rule that can generate the possible paths that it can take from one destination to another or from one train station to another. We utilized the maze problem prolog code that we have done in class as a reference for our rule. Initially when we were declaring the rule we encountered a problem with infinite loops, the same problem that would have occurred for the maze challenge. We later resolved the problem by keeping a record of where we have been, the same solution for the maze challenge. The successfully developed path generating rule in our program is called travel. The rule essentially takes a starting train station and a final train station that the user enters and generates all the possible trains and paths that the user can take from their starting position to their destination. After we have developed the travel rule, we decided to develop a shortest Path rule that can give the user the fastest itinerary to travel from one locator to another. To generate a shortest path we added weights to our train paths. For example, in our program train A has a path from station 4 to station 5 and a path from station 5 to station 6. We gave both paths a weight of 1 so if we want to travel from station 4 to station 6 on train A, the total weight of the path would be 2. Using this concept of adding weights to our paths we can identify the shortest path if a path has the smallest total weight. The next thing that we implemented to our program are the states of the trains. This aspect was implemented for the belief revision function of our program. Depending on the state of a train, the weight of the train's paths can change and thus the shortest path might also be altered. Currently, we have the states of clean, normal, and dirty for the trains. If the state of a train is "clean" then the weight of all its train paths would be 0.5, whereas if the state of the train is "normal" then the weight of its train paths would be 1, and if the state of the train is "dirty" then all its paths would have a weight of 1.5. In the end, we added a user interface to our program so that the user can interact with our program. Our program, as of now, does really well in generating the possible routes to

take given the starting train station and the ending train station, and it is fast in producing the shortest path route. In addition, we can easily adjust the train model of our program such as adding or removing trains and train stations and it would still work perfectly fine. The drawbacks that we can identify from our program right now is that it can only update 1 train state at a time, no permanent closing down of stations, and we are not sure if our program can handle large and complex networks. We plan on improving our program by adding more trains and stations to our network as well as more states that the train can be in, such as delays or permanent shut down.

Conclusion

In conclusion, we have completed a model that provides commuters with the shortest route to the target station and allows commuters to decide whether to take the train or not depending on the condition of the train. one of our drawbacks, it can only update 1 train state at a time, ,it allows to update all train conditions at once. However, even if our program works normally, it's unclear if this will work in a complex or huge network.