Kuncheng Feng, Taeyoung Park, Johnson Liu
Professor Daniel Schlegel
CSC366 Project Task 9
11/16/2021

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 `shortestPath` 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.