Jiayi (Josh) Wang jw6dz
Kevin Qian kq4hy
Homework 3
CS4710

Analysis Document

Requirements

- Describe the algorithm you implemented. Why does it make sense to play the game this way? Is this the way a human would play, or is the strategy fundamentally different? Did you need to use heuristics, if so what did you choose and why?
- Describe in detail how you tested your strategy. How did you detect issues with your AI's approach? Give me some specific examples of testing you did and what these tests helped you learn about your approach to the game.
- Give me some data on how well your AI plays against some baselines. Maybe implement a quick random AI and see how well your implementation does. Analyze your results and discuss. If you tested against others, give me some data on how well your AI did. Where does your code fail and/or succeed? Why is that the case?

Algorithm

The basic algorithmic approach that we took was to manipulate the Markov Design Processes (MDP) to fit Ticket to Ride's purposes. MDP is a model that contains:

1. Set of possible states, *S*.
2. Set of possible actions, *A*.
3. A reward function *R(s,a)*.
4. A description *T* of each action's effects in each state.

Similarly for this AI, which we affectionately named Skylar, those same four attributes were used with the following definitions:

1. Set of possible states would be either *buy-able* or *un-buy-able*.
2. From the *buy-able* state, there would be two actions, able to claim more than one route (a1) and only able to claim one route (a2).
From the *un-buy-able* state, there would be three actions, draw a new train card and then use claim a route (a3), draw a new train card but hold off on buying for better future investments (a4), and draw two new destination cards (a5).
3. The reward function is the number of points that the AI can receive based on the action that you take. The reward function for a1 is the number of points that the player gets for claiming that route as well as all subsequent routes that it can claim. The reward function for a2 is the number of points that the player can receive for claiming that single route and then going back to the sate of *un-buy-able*. The reward function for a3 is the amount of points that the player receives from

claiming a route after drawing a train color card. The reward function for a4 is the amount of points that you can get if you save up a specific color instead of buying immediately. The scenario for this is, if you have 2 yellow cards you can build on a route where color does not matter but if you save up for a larger route where yellow is required, you'll receive more points. The reward function for a5 is threshold of 15 - sum of current points from the destination cards. This will ensure that if the player starts with less than 15 points, and thus, easy to accomplish routes, more destination cards will be added to get more points overall.

4.

a1: buys the most valued route it can afford, stays in the buy-able state because even after that buy, it can still afford another route.

a2: buys the most valued route it can afford, resulting in not having enough resources to buy another route. Moves to the un-buy-able state

a3: draws either a rainbow card from the revealed stack or a random card from the deck which would put prepare it to claim a route on the next turn. Moves to the buy-able-state.

a4: draws either a rainbow card or a random card from the deck, but can't afford a route still so it stays in the un-buy-able state.

a5: draws two destination cards, stays in the un-buy-able state.

Since destinations are essentially points and edges, Djikstra's algorithm was implemented to calculate the shortest path between the two destinations and an array of Routes was returned. By calculating the shortest path for both Destination cards, overlapping paths are given more priority because the player can receive more points if they claim those routes. If a path in the shortest path is claimed by another player, Djikstra's will be called again to calculate the shortest path with taken path removed from the list of nodes and edges. The MDP will be called on the entire system but it specifically evaluates the priority list of the routes that were calculated by Djikstra's.

While the algorithm is relatively basic, it accomplishes a wide variety of things. The most basic activity is, what happens when the player does not have enough Train cards to claim a route? A human will choose to pick up either a train card or two new Destination cards. Similarly, the AI will take into consideration if it is better to pick up Destination cards or draw a train card, based on the reward function. The logic behind drawing Destination cards is if the player has Destination cards that are worth a total of less than 15 points, then it makes sense to draw more Destination cards to find more overlapping Routes because this allows for planning for an overall of more points. The implementation immediately does not consider drawing more Destination cards if the player already has four or more because drawing too many will result in a large loss of points at the end if none of them are completed. When the AI decides to draw, two options must be considered. One, what is the reward if a Train card is drawn and a route is then "claim-able"? And two, what is the reward if a Train card is drawn and claiming is held off for more turns? This is similar to human players as well because when humans play, they must consider if it's more advantageous to build immediately or to stock up on certain colors to build an extremely long path in the future. The next activity is what happens when the player owns enough Train cards to claim a route? A human will analyze the amount of points that they can get if they use up all possible Train cards to claim routes and the amount of points if they just use a portion of their Train cards and then go back to the drawing action. Similarly, the reward function for the AI will compute if it is more beneficial to build one route and then go back to the state of *un-buy-able* or stay in the state of *buy-able* and continue to claim routes.

By playing the game in two states of *buy-able* and *un-buy-able*, points can be accumulated based on the limited actions that exist within this process. Furthermore, this implementation is easy to calculate the reward and will, in most cases, claim routes that generate the most points so even if Destination cards are not completed, the final number of points will be greater than 0 due to the nature of the AI.

At each turn, the AI will evaluate the rewards that would follow from going with any of the actions it can take given the state it's in. The rewards calculation functions take into account future steps and rewards the AI can reap if it followed the action at the current turn. After rewards calculation, the AI takes the action that generates the greatest reward and moves to the state according to the action. This continues until the either of the players hit the minimum game pieces cap.

Testing

- Describe in detail how you tested your strategy. How did you detect issues with your AI's approach? Give me some specific examples of testing you did and what these tests helped you learn about your approach to the game.

We extensively tested our strategy as we developed it, thus catching issues and optimizing to improve the AI as we progressed. We manually tested our AI for the most part. First, we tested it against the human player that only drew cards to see if Skylar was correctly assessing the routes he needed to claim to finish his destination cards. While doing this, we found that we should prioritize getting the longer/colored routes over the smaller ones, thus refining our a1 and a2 actions/algorithms. Then we tested to see how Skylar reacted when the other player claimed routes that Skylar needed, forcing Skylar to reroute in order to finish his destinations. Skylar will work with the other player's choices and still try it's best to complete its destinations. One thing we noticed during testing is that if Skylar's first two destination cards are both relatively low-valued, it was more efficient for Skylar to pull two more destination cards at the beginning rather than finish the initial two it draws before drawing two more destination cards. We set the required sum of destination rewards at the beginning to be 15. If the first two cards drawn initially don't add up to 15, we immediately draw 2 more destination cards at the beginning. Since we place greater value towards shared routes, having 4 at the beginning to evaluate increases the chances of having shared routes, thus making the overall path-finding and claiming algorithm more efficient.

Data Analysis

Give me some data on how well your AI plays against some baselines. Maybe implement a quick random AI and see how well your implementation does. Analyze your results and discuss. If you tested against others, give me some data on how well your AI did. Where does your code fail and/or succeed? Why is that the case?

We made several improvements to our AI as we tested and worked on Skylar. At first, when we looked ahead in our a1 and a3 states, we didn't consider the value of grey tracks or the value of rainbow cards in our hand, prioritizing colored tracks over grey tracks. That threw off our a3 Q-values in our un-buy-able state, resulting in an average of ~28 turns passing before Skylar was ready to claim routes. After observing that while testing, we realized that that's very unrealistic, and that Skylar should try to buy earlier. We re-evaluated our code that handled buyable paths evaluation and decided to order the routes that we needed to claim to meet our destination card requirements by the cost to purchase the route, with colored tracks having precedence over grey tracks. This is a viable decision since colored tracks are in general longer than grey tracks with one-cost routes only being grey tracks since they're easily obtained (low priority). By ordering the list of buyable routes, we can evaluate claiming them with a focus on the longer/more specific routes and our look-ahead for a1 would account for that as well, being able to loop through the buyable routes. We did not have the chance to test it against other AIs.