The Difficulty of Learning Ticket to Ride

Daniel Dinjian & Cuong Nguyen

1 Abstract

Ticket to Ride is a very popular, award-winning board-game where you try to score the most points while building a railway spanning cities in America. For a computer to learn to play this game is very difficult due to the vast state-action space. This project will explain why featurizing your state, and implementing curriculum learning can help agents learn as state-action spaces grow too large for traditional learning methods to be effective.

2 Introduction

For our project, we will be teaching an agent how to play the popular board game "Ticket To Ride" (TTR). The main goal of TTR is to score more points building your railway than your opponents. Points are awarded immediately for buying routes for your railway, and at the end of the game for having the longest contiguous railway and having a railway that connects specified cities.

This game's scoring system makes it a very interesting problem because it places importance both on maximizing immediate rewards in building routes, and on planning for the end game by prioritizing future end-game rewards. Before talking further, I'll define the relevant rules.

2.1 Relevant Rules of playing TTR

2.1.1 Game Setup

The TTR board is essentially an undirected graph where nodes are cities and edges are routes connecting two cities. Routes have varying lengths and colors which correspond to the cost of purchasing them. There are two supplemental decks: a deck of colored rail-cards used as resources for purchasing routes, and a deck of unique goal-cards specifying two cities and a reward for connecting them via your railway. If you fail to connect the cities specified on one of your goal-cards, you lose that many points as punishment instead of getting the reward.

Each player starts with 45 train pieces, 4 rail-cards, and three goal cards of which at least two must be kept. When building routes, not only must you pay the appropriate number of colored rail-cards, you must also place that many

train-pieces on the board. Whenever anybody runs out of train pieces, gameplay ends and final scoring is done.

The game supports 2-5 players, so agents trying to complete their individual goals may have overlapping routes. As a result, to succeed, each agent will have to optimize their planning to complete their goals while preventing antagonizing agents from sabotaging their route.

Agents' conflicts of interest are:

- Each route can only be bought by one player.
- Players can only place a certain number of train pieces before the game ends.
- Rail-cards are discarded after use, the rail-deck isn't reshuffled until it empties.

What these conflicts mean for the agent is that paying attention to your opponent's behavior is important to success because you're competing over limited resources.

2.1.2 Game Play

The game moves in a round-robin manner where each agent takes a turn in a predetermined order. During each agent's turn, it has the option to do 1 of the following 3 actions:

- **Collecting Rail-Cards:** To prepare for the building costs of buying routes, an agent may use its turn to draw two rail-cards. Rail-cards are either drawn randomly from the deck, or from a pool of 5 face-up cards.
- **Building Routes:** To buy any route, an agent must use (and discard) the required rail-cards from their hand. Then they place their train pieces on the route both to demonstrate ownership and to keep track of how many train pieces they have left.
- **Drawing New Goal-Cards**: An agent may draw 3 additional goal-cards and is required to keep at least one of them. This is a high-risk high-reward action because new goal-cards could be easily accomplished or could be incompatible with your previous strategy.

2.1.3 Scoring

As mentioned, the objective of the game is to collect as many victory points as possible. Score is calculated as follows:

1. **Building Routes:** Whenever an agent builds a route, they are awarded points immediately. The amount of points is based on the number of tracks required to connect those two cities. See Table 1.

2. **Meeting Objectives:** Goal-cards specify objectives for agents to build railways that connect two cities. Each goal-card has some predetermined score (5 to 23) based on the approximate difficulty/lengthiness of connecting those cities. At the end of the game, if the agent has met that objective, they gain the corresponding number of points. If the agent has not met that objective, they lose that many points instead.

3. **Longest Road Reward:** At the end of the game, the player with the longest contiguous railway is awarded 10 points. The longest contiguous railway refers to a sequence of routes that doesn't backtrack or jump from city to city.

Route Length	Points
1	+1
2	+2
3	+4
4	+7
5	+10
6	+15

Table 1: The immediate points gained when an agent builds any route based on route length

2.2 Hypothesis

Ticket to Ride is a very complex game for a computer. It's relatively easy for people to learn because we can plan in advance and calculate the value of each move, albeit subjectively. However, if a computer were to try and use traditional reinforcement learning methods to play this game, it would require a state for each combination of goal-cards, hands, boards, etc. which is clearly infeasible. Not only would it be unreasonable to attempt to store the value of each state or action, even if that was done, the time it would take to train the model would be even more ludicrous. Simply stated, it's computationally intractable.

Therefore, learning Ticket to Ride requires the artificial development of new features based off of the environment that are imbued with relevant information. Feature engineering is an incredibly powerful method of using preordained field-knowledge to tell models what environmental factors will be important to them accomplishing their goals. In environments with small environments that can be fully explored the use of feature engineering may be limited, but in problems such as TTR where the state-space is too vast to explore, feature engineering will be the single most important factor in creating a model that can learn.

In addition to having good feature engineering, another method of helping agents explore vast environments is through curriculum learning. In curriculum learning, rather than placing the agent immediately into their environment, they learn to perform portions of their task well in mini simulations. By learning in

these controlled environments, the agent will get exposure to the important tasks it has to accomplish in the full environment rather than taking a much longer time to discover them on its own.

Although ultimately we were unable to construct an agent that successfully learns to play Ticket to Ride, we are confident that the theoretical model we'll describe would be able to learn Ticket to Ride given enough time. We'll also analyze why we were unsuccessful and how future work could avoid some of the technical problems our model ran into.

3 Background and Related Work

In Chapter 9 of Sutton & Barto's Reinforcement Learning: An Introduction [2], the concept of feature engineering is first discussed. The way Sutton & Barto introduce feature engineering is motivated by linear methods. Theoretically if we have all the information pertaining to an environment, some combination of that information will be sufficient for determining how to act in any situation. However, we don't necessarily know what shape that combination will be or what type of function will best learn it. On the one hand a neural network can approximate any function, but on the other hand if we construct our feature space to be linear then a simple linear method will work fine. Sutton and Barto explain the benefits of having, or constructing, a feature space that lends itself to linear methods. It's conceptually straight-forward, the features we consider are easily modified, and it effectively outlines what aspects of the environment are important for the model to consider giving the model a big head-start.

Given the complexities of the TTR tasks, it's quite likely that our agents would gravitate towards prioritizing short-term value of building routes over long-term priorities like meeting objectives and building contiguous paths. Learning through exploration would most likely take an extremely long time to fully explore the environment even after the state-space has been featurized. In response, we will introduce a curriculum of sub tasks to aid novice agents in their learning of the game objective. Our approach will be similar to the approaches shown by Narvekar et al.[1].

4 Problem Formulation and Technical Approach

Our technical approach to teaching an agent to play TTR was first to create an environment where we could simulate gameplay. We then featurized that environment and fed it to our agents to play games and learn as follows. However, our agent never really learned due to both technical and design issues. In the following formulation, a proper design for learning TTR will be described as well as an argument for correctness.

4.1 Environment Setup

The first step before learning to play TTR is to have a simulation of the game implemented. This simulation should track the true environment (player hands, scores, train pieces, goals, board, etc). And should be displayable by a GUI for purposes of observing your model.

Next create a portal for your agents to play the game. Generate an episode and pass in agents. At this point, agents should simply be passed the environment at their turn, have access to their possible actions, and choose randomly. Before we move on to having the agents learn, ensure that you can simulate episodes, or games, start to finish between your models.

4.2 Featurizing the Environment

In order for our agents to learn anything, we have to featurize the environment into a format in line with Sutton and Barto's linear method recommendation [2]. The environmental factors that we want to highlight in our constructed features have to be the ones that we use to determine which actions will maximize our net reward.

- **Available Build Routes:** Each route is first-come first-serve. Agents need to prioritize building routes based on what routes are still available and how important they are to accomplishing agent's goals.
- **Drawable Rail-Cards:** Within the deck of track cards, there are 6 railroad track types that exists in limited numbers. Within a turn, an agent can either draw from the top of the face down deck, or pick up from the 5 face up replenish-able community pile, or do both. The strategic trade-offs between drawing from the deck or community pile lays within the realm of secrecy and draw probabilities.
- **Owned Rail-Cards:** At any time step, an agent must optimize its use of in-hand resources given the current state. Given the specified build expenses to certain routes and competition for claiming routes, it may be more advantageous to use an in-hand resource sooner rather than later, or vice-versa.
- **Remaining Train Pieces:** When either you or your opponent runs out of train pieces the game is over. Furthermore, if you waste too many train pieces then you may not be able to finish meeting your objectives.
- **Approximate Distance to Meeting Objectives:** Based on the expected number of routes, rail-cards and train pieces left after you've met your objectives, your best option may be to draw new goal-cards.
- **Antagonist Agents' Previous Actions:** Observing the history of competing agents such as what cards they've collected or where they've been building can inform the protagonist agent's update of action-state values.

By passing this information to our agent, it will be able to learn how these factors correspond to determining the agent's optimal action. Note that an agent's action-space is equivalent to the options mentioned in Game Play **2.1.2**.

4.3 Determining Reward

Although we now have features that will help our agent to evaluate what the state of the environment is, there's one more step required for learning. The reward.

In playing TTR, our agent's main goal will naturally be to win, sowe need a reward function that rewards agents for winning. Agents win by having the most points at the end of the game, so we make the obvious connection of rewarding our agents equivalent with how they score points. See Scoring, **2.1.3**.

However, this reward method actually rewards agents even if they don't win, which we do not want. Therefore, at the end of the game we introduce one more reward for agents. The winning agent keeps their reward equal to the number of points they score, but the losing agents all get a penalty equal to the winning score. Therefore the net punishment of an episode is scaled according to how poorly the agent performed.

4.4 Curriculum Learning

Now that we have all the components our agents require to learn, we should be able to let them play a few thousand games and see what they've learned. Agents will play, they'll win or they'll lose and their value functions will update to help them prioritize the right moves based on the featurized representation of the environment. Having gotten to this point you would even be able play against your agents and see how they perform. But chances are, even after learning for a long time they wouldn't play very well.

The reason that agents would be learning slowly is because they're exploring naively. They would only be playing against other agents, exploring the environment randomly and using the strategies that have worked for them. Even after playing thousands of games it's still quite possible that they wouldn't have developed nuanced strategies simply due to lack of exposure to certain circumstances that only come up between experienced players.

Exposure to these important circumstances is why curriculum learning is so important. You can train an agent to be good at a specific subcomponent of its job so that when it encounters that aspect of its job in a real environment it will remember how to act. As the curriculum progresses, the agent is introduced to more and more of its overall goal until finally its ready to explore its true environment.

Our curricula will primarily consist of learning three sub-goals.

1. Playing the game individually, drawing cards and building train tracks to maximize score.

- Agent will be alone taking turn after turn, trying to maximize average points per turn.
- Agents will learn to make educated decisions about when and where to build trains.
- Agents will learn when and where to draw cards from.
- Agents will learn how to build a railway that connects their goal cities.
- Agents will learn how to connect routes across multiple cities to score longest road.

2. Accounting for additional opponents

   - Agents will compete in this sub game with just a single goal-card
   - Agents will learn to consider their opponent's potential moves when drawing resource cards or building tracks.
   - Agents may learn or be encouraged to sabotage each other.
   - Losing agents will have their end game rewards stripped away and penalized by point differentials

3. Optimistic Initializations

   - Agent may be placed in situation where game has almost ended and agent has a variable number of turns left to complete assigned tasks.
   - Agent may be playing alone, assigned to meet goal-card objectives, or against other agents assigned with making strategic end-game choices to win.
   - Agent will receive reward or penalty based on the completion of their assigned tasks.
   - Environment may end prematurely to ensure urgency in agent's strategic approach.

After completing the curriculum, the agent is considered "educated" and we expect educated agents to make wise decisions in the real environment. Knowing when to draw cards, where to build routes, and to complete their goal-card objectives better than naively trained agents.

Going through this curriculum helps agents to develop a proper roadmap for playing their way through real games. Having this roadmap allows agents to use their exploration finding nuance and perfecting their overall strategy rather than spending all their time piecing together strategies that may or may not be effective.

5 Expected Outcomes

There are a few different types of agents posited in our technical approach, unfortunately, we do not have actual results for any of them. However, we will explain why they should have worked and how they would have been expected to perform so that future implementors can learn from our work.

5.1 Unfeaturized Model

If we were to attempt to pass agents all the information in our environment and just hope for the best, that would not work. There are approximately 100 routes on the board, many many combinations of possible board settings. Having to retrain that for the countless possible hand settings, and then also for the different possible goal cards in an agent's hands.. Hopefully it's clear to everyone that this is an unreasonable waste of time and who knows if this model could be trained in a human lifetime.

5.2 Featurized Model

Featurizing this model's inputs is exactly what would make learning possible. By featurizing the inputs we create a generalized state representation that frees our model from having to relearn value functions in every combination of similar circumstances. We could construct a learning curve for this model and we would expect it to very slowly learn how to play.

The major issue with pure learning on featurized state-representation is that we'd expect to get stuck in many local optimal policies that are far from as good as the global optimal policy. This would happen because the agent would be exploring naively, it would try out various actions and quickly start making a strategy based on the random actions it had tried.

It would be like creating a path through a forest. At first you cut your way through the forest randomly, but after you've created a path, future trips through the forest will use the same path only deviating slightly where the path is clearly poor. Even if the entire trail being made is suboptimal, once it exists it becomes much harder to rediscover a whole new path. And similarly the naively exploring agent, once it has created an approximate strategy, will have a very hard time finding entirely new strategies and will instead try to perfect its strategy even though that strategy can only be improved so far.

5.3 Curriculum Learning

Even though featurized models are able to learn, they're very prone to getting stuck in suboptimal local policies due to their naive exploration of the environment. Curriculum learning slowly introduces the model to a preordained best strategy by having it learn first in controlled environments.

This process helps the model figure out approximately how to traverse the environment so that the strategy it perfects is as good as possible. As such,

we expect educated agents that have gone through the curriculum to play TTR much better than naive agents. Not only will they finish the curriculum playing better than naive agents, they'll also learn faster and converge to a higher level of playing with a more nuanced strategy than naive agents.

6 Conclusion and Future Work

Unfortunately, the technical work done for this project failed to yield the expected results. This was due to a combination of poor design and poor implementation early in the process. Therefore in this report I've attempted to describe a stable pipeline with a good, modular design that will allow readers to easily implement this project in the future.

The main goal of this project was to demonstrate how to approach problems where the environment is too vast to be directly learned by a model. Hopefully the two major approaches are clear. It's integral to featurize your environment, especially constructing features that lend themselves to linear methods so that the agent will be easily able to determine optimal actions. However, while this allows the agent to learn to operate in the environment, there's no guarantee that it will learn an optimal policy. To address this, designers can take alternate methods of giving their models exposure to optimal strategies for succeeding in their environment, either through curriculum learning or alternate approaches such as transfer learning or learning policies from human teachers.

Naturally future work includes fully implementing this project and demonstrating the accuracy of the claims made here. But beyond that, future work could also include comparing various featurizers, closer observation of the adversarial nature of competing agents, cooperative learning where agents share their experiences, or other similar perturbations of the experiment as its described here. Hopefully having read this paper you are ready to implement your own agent to learn Ticket to Ride, and have also learned about how to create agents for learning in vast environments.

References

[1] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[2] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.