Reinforcement Learning in Public Transportation

Bonnie Hu
260556970
guanqing.hu@mail.mcgill.ca

Andrei Chubarau
260581375
andrei.chubarau@mail.mcgill.ca

Haydar Talib
119829250
haydar.talib@mail.mcgill.ca *

Abstract

In this work, we explore the use of reinforcement learning (RL) in the context of public transportation optimization. We create a dynamic Gym-like (1) environment to model real-life public transportation networks and apply several RL methods to validate its learning potential. Our results demonstrate the effectiveness of the DynaQ algorithm under our environment and emphasize the explosive influence of model complexity.

1 Introduction

Urban planning in public transportation is a complex domain, and optimal solutions for creating and operating transportation routes are non-obvious. Many variables should be considered in developing a public transportation network, including (but not limited to): population density layout, commercial centres and office building locations, essential services locations, size of the population, daily traffic and causes for its variation, and so on.

On an individual, human scale, passengers in large urban centers can commonly be impacted by delays and slow progression while using various available public transportation options. Many factors can contribute directly or indirectly to the delay a person may experience on their way to work, an urgent appointment, or their family. Such factors can include sudden or gradual traffic changes due to, for instance rush hour, construction, and weather. This can also be caused by non-optimal public transportation scheduling, which may simply be inadequate and unable to keep up with the changing demand. Perhaps a more flexible, non-static scheduling that is responsive to most current conditions may be introduced to improve the situation.

Reinforcement learning (RL) is a domain of machine learning that concerns itself with sequential decision making with the ultimate goal of making the best possible, i.e. optimal, decisions given current conditions. As part of the current project, our goal is to utilize RL methods in the context of public transportation to come up with efficient policies for deploying, or in other words, scheduling, public transport.

This work introduces an environment for testing and analyzing reinforcement learning algorithms towards the public transportation optimization problem. The environment is Python-based, structured similarly to the environments provided by the currently available and popular Pycolab (2) and Gym (1) packages. Our final goal is to adapt and release our environment under Gym.

To focus down the vast domain of public transportation in urban planning, our environment simulates a bus route with varying and configurable conditions and characteristics. We model a time-dependent environment in which traffic and transportation demand varies over time.

*Submission for the final project of COMP 767 - Reinforcement Learning

Once we formulate the environment in this manner, it becomes clearer how this problem may be suitable for reinforcement learning-based approaches. Buses operating on a predetermined route, interacting with the environment through picking up and dropping off passengers at different stops, form a structured behavior. This behavior can be tied to positive and negative feedback as passengers are taken to their destinations or otherwise kept waiting.

More details on the relationship between the bus route problem and a reinforcement learning formulation are provided in Section 2. Although this environment may appear conceptually simple, we will show how complexity rapidly emerges in simulating static and variable traffic conditions, the rates of passenger arrivals at the different bus stops, their different destinations, etc.

In Sections 3 and 4 we describe our results of using the Dyna-Q algorithm (3) and its performance in attempting to determine an optimal policy under our bus route environment. In Section 5 we continue the discussion on how this environment can be further extended and where it might integrate or otherwise fit into the landscape of public transportation optimization, urban planning, and reinforcement learning in general.

1.1 Related Work

To continue our focused view, we look within the reinforcement learning field as we explore prior works. The formulation of related problems in route finding and optimization is sometimes expressed as a Markov Decision Process (MDP), and such problems include transit trip planning based on existing public transit options (4; 5), as well as bus rerouting problems (6). Our controlled environment approach will find analogs in literature, for example the work on multi-agent systems for bus and traffic light optimization by Tlig et al. (7). In this latter work, however, the environment is created in a proprietary commercial software, which may make extensibility and reproduction of work challenging and stifling to progress in this domain.

2 Creating a Bus Route Simulation for Reinforcement Learning

Our proposed environment operates a "tick"-based day-night cycle simulation and models a simple public transportation network with varying demand. The length of a tick is configurable; in our work, a tick is set to simulate 5 real world minutes. Passengers have randomized destinations and dynamically "arrive" to stations; similarly, traffic conditions may change over time.

The environment is updated at every tick: buses move between stations, passengers come and leave, etc. Traffic conditions incur delays between stations, which implies that a bus may take several ticks to move between adjacent stations. A single action can be played by the environment at every simulation tick. The system can be configured to operate in online or episodic mode. In the later, several definitions of an episode are possible: i) the time it takes to deliver K passengers, or ii) K hours of simulation time, where K is a user-defined constant.

The bus route is circular as illustrated in Figure 1 with a configurable number of stations and passenger distributions. We define a single terminal station in the route (Station 0). When a bus arrives at the terminal station, it unloads all the passengers and vanishes. If needed, the environment can also be adjusted such that the unloaded passengers would wait for the next bus to pick them up.

2.1 State Representation

Our environment provides several ways to represent states with various levels of complexity as can be seen in Table 1. States are represented as strings; the complexity of state representation is configurable. For each station, we may include information about the current number of passengers and buses; multiple stations get concatenated to create the complete state string.

The simplest state representation considers each station as a single Boolean (true if there are passengers at station, false otherwise), and encodes the station of the network as a binary number, where each bit corresponds to a station. Information about deployed buses can also be included. This can be stored as a Boolean for every station.

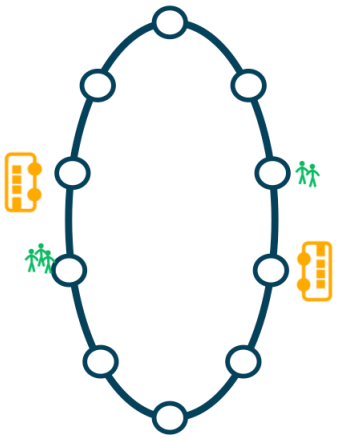

Figure 1: Visual representation of the bus route environment

Table 1: State Representation Examples

Method Description	Complexity	Example State String
Boolean passenger count	Low	"(T)(F)(T)(F)(F)"
Single numeric bus count + numeric passengers	Medium	"b5:(32)(0)(19)(2)(0)"
Numeric bus and passenger counts	High	"(32,1)(0,3)(19,0)(2,1)(0,1)"

Traffic conditions can get incorporated by further insertions into the state string. Several simplifications can be applied to reduce the state space, for example encode solely the total number of buses in the network versus encode number of buses per station.

So far, state encoding ignored the accurate number of passengers and buses at the stations. As more and more information is included in the state string, the state space increases. Typically, under binary encoding, for k types of information and n stations, we have a finite state space with n^k states. On the other hand, if accurate numbers are used instead of Boolean values, the state space becomes infinite as k is not bound. State representation accuracy can thus be traded for higher dimensionality.

More complex representation provides the agent with more information about the environment that might be useful to learn a finer policy. However, it also leads to larger state space for the agent to discover and learn, which naturally results in slower learning convergence or even failure to do so. The choice of state representation depends largely on model complexity, learning algorithm, and other specific requirements.

2.2 Action

Actions define how the agent interacts with the environment. We define three main categories of actions. The agent can choose between: i) do nothing, 'wait', ii) send a bus from the terminal station, 'schedule', and iii) send a bus from a station other than terminal station, 'rescue'. The rescue action is intended to be taken when there is a traffic jam causing the "regular" schedule to be inadequate: passengers are waiting longer than usual. Sending a rescue bus injects a bus into the network to pick up the angry passengers and adapt to a bad environment. Thus, a typical action set can be $[' wait',' schedule',' rescueFromStation5',' rescueFromStation10']$, where rescue station number is configurable.

2.3 Reward

As the ultimate goal is to transport as many passengers as possible to their destinations while maintaining a balance between customer satisfaction and network maintenance, we incorporated these factors in our reward function. Customer satisfaction is modeled with regards to their waiting

time, i.e. the time a passenger spends at a station before entering a bus. Short and long term effects of network maintenance are approximated by an instantaneous penalty at the time of bus deployment and an ongoing tick-based reward penalty linearly related to the number of deployed buses.

More specifically, our reward function consists of instantaneous and ongoing components. One unit of instantaneous positive reward is given for each delivered passenger. Furthermore, K (configurable) negative units are given for each newly deployed bus. Ongoing rewards are applied at each simulation tick. We add one negative unit for each currently deployed bus, as well as for each passenger currently awaiting a bus.

These reward rules encourage the agent to transfer passengers faster, but also balance the number of buses by deploying these more efficiently. Fine tuning the parameters influencing the reward function was a major concern. Overall, the combined reward at each time step is typically negative, thus cumulative reward throughout the episode is an increasingly negative value.

3 Reinforcement Learning Approach

We tested the proposed environment with the DynaQ algorithm (3), because it is a powerful learning algorithm that is easily implementable, capable of planning, and applicable to on-line problems. DynaQ learns two main models: the state-action value function $Q(s, a)$ and the environment model $M(s', r|s, a)$. Dimension of Q increases linearly with number of state, while dimension of the environment model increases by power of 2. When the state space gets larger and larger, DynaQ becomes slower in retrieving data from the models and could be the main limitation of the algorithm. Another limitation is the assumption of a static environment, which means that the model learned by DynaQ gives state s and action a always results in the most recent next state s' and reward r. Since the passenger distributions can change, the environment is not static any more, which makes this assumption a limitation during learning.

A lot of effort was spent on defining the used state representation and actions sets, as well as fine-tuning the reward function, with the goal of providing a combination that results in more efficient learning. We started with simple scenarios and gradually increased the environment complexity in order to investigate how challenging that is for DynaQ. Parameters of DynaQ (e.g. planning steps, epsilon) were also fine-tuned to achieve faster converging rates. A simple test-case consists of an environment with 10 stations, 2-3 actions, static passenger distributions, and low-complexity state representation. A more challenging environment has more stations, various rescue actions, dynamic passengers and traffic, and state representation that contains detailed environment information. Test results are presented section 4 which follows.

4 Results

Since DynaQ managed to handle all the simple environments successfully based on our tests, the testing results below comes from a medium-complex scenario consisting of 10 stations, 3 actions ('wait', 'schedule', and 'rescueFrom5'), and rush hour simulation. Rush hours were defined at 6am and 6pm when passengers suddenly increase. We also introduced a sever traffic incident at episode 250 in order to see how fast the agent can recover from changed environment. Figure 2 presents the test results. It is easy to see the convergence of the algorithm after the first 90 episodes. When we look at the optimal policy at episode 200, the agent actually learned to send bus occasionally similar to what a regular bus schedule does, but at the same time, it also knows when to send a rescue bus to reduce the passenger complaints. At episode 250, the severe traffic incident was introduced whereby a large delay was added in between two stations. This can be observed as a dip in total reward, but eventually the system recovers by episode 500.

5 Conclusions and Future Directions

In the current project, we investigated the use of reinforcement learning in the context of complex transportation systems modeled by a Gym-like custom environment. Several existing learning algorithms were used to test the implemented environment. Our results provide insight into the complexity involved in the macro-optimization of scheduling and deployment of public transport and demonstrate that even the simplest of configurations are challenging for learning. Further work

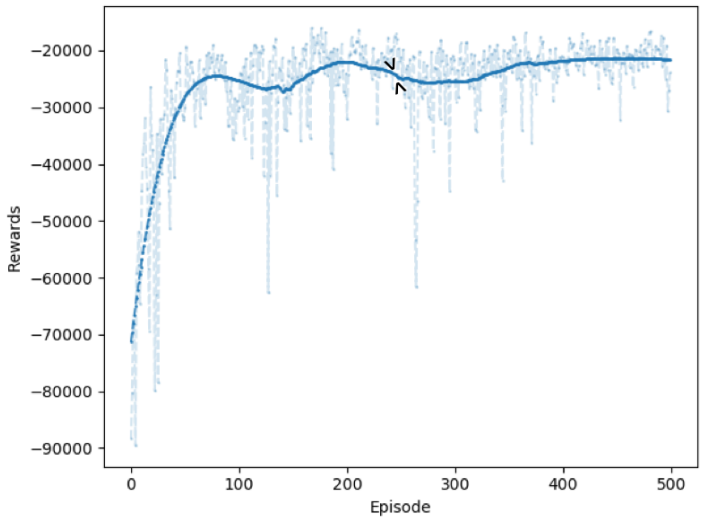

Figure 2: Dyna-Q performance during learning as given by the total cumulative reward in simple traffic environment. Total reward clearly improves within the first 200 episodes.

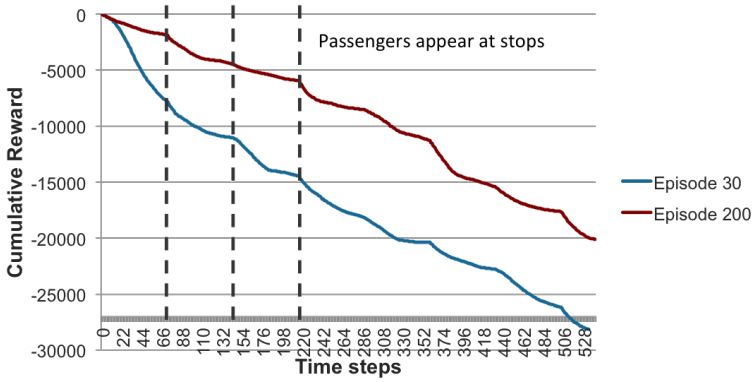

Figure 3: Cumulative reward of runs given by DynaQ after 30 and 200 learning episodes. DynaQ-200 shows a marked improvement over DynaQ-30, as the cumulative reward is significantly higher. On both curves, it can be observed how key moments of an episode impact cumulative reward, in particular when the passengers configuration changes at equal intervals.

includes finalizing our environment under Gym, implementation of multi-route bus networks, as well as other general quality of life improvements.

5.0.1 Author Contributions

All three authors contributed equally to all aspects of the project, working in tandem throughout.

References

[1] OpenAI Gym. `https://gym.openai.com`

[2] Pycolab. `https://github.com/deepmind/pycolab`

[3] Sutton, R., Barto, A. (2017) Reinforcement Learning: An Introduction *MIT Press, Second Edition*

[4] Nuzzolo, A., Comi, A. (2017) A normative optimal strategy in intelligent transit networks *Transportation Research Procedia* **27**:380-387

[5] Boyan, J., Mitzenmacher, M. (2001) Improved results for route planning in stochastic transportation. *Proceeding SODA '01 Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms* :895–902

[6] Rouhieh, B., Alecsandru, C. (2012) Adaptive route choice model for public transit systems: an application of Markov decision processes *Canadian Journal of Civil Engineering* **39**(8):915-924

[7] Tlig, M., Bhouri, N. (2011) A Multi-Agent System for Urban Traffic and Buses Regularity Control. *Procedia Social and Behavioral Sciences* **20**:896–905