Dynamic Prediction and Control for Bus Bunching

Abdullah Aldamer * Samuel Bobick † Soen Flochlay * Elouan Pulvéric *

1 Introduction

Bus bunching is when buses on the same route arrive at a stop very close together. This is often catalyzed by traffic congestion, slow boarding or alighting of passengers, and differences in bus speeds. As a bus falls behind schedule, more passengers congregate at each bus stop, causing boarding to take even longer to board those passengers, causing the bus to fall further behind. Conversely, if a bus runs ahead of schedule, less passengers accumulate at each stop, and the bus gets further ahead of schedule. These dynamics lead to bus bunching. Bus bunching can lead to longer wait times, overcrowding, poor reliability, and higher operating costs for transit agencies.

A new technique involves using real-time data to monitor bus locations and adjust schedules by holding buses to maintain equal spacing and keep wait times consistent, a control scheme which is commonly referred to as dynamic holding. However, with many buses on the same line at once, holding a bus to alleviate bunching in one part of the line can create bunching bunching in previous parts of the line unless this action is coordinated with other buses. In this paper, we employ neural networks to forecast bus bunching and design a control strategy with reinforcement learning.

2 Literature Review

Below we present prior work on holding strategies. Both classical controls and reinforcement learning techniques have been applied to this problem. Our work differs from the below works because its predictions and actuations are derived *solely based on bus location data.* This is useful to bus operators in cases where demand data is inconsistent, unavailable, or not tracked.

2.1 Linear Control Dynamic Holding Strategy from Virtual Schedule Fitting

Many high frequency bus lines operate on promised headways instead of fixed schedules. However, one approach relies on fitting headway-based bus lines onto virtual schedules [Xuan, 2011]. This approach uses bus arrival deviations from a virtual schedule to maintain regular headways to formulate a linear control problem. When a bus arrives early or late, it is held at the station to maintain a regular interval between buses.

2.2 Multi-Agent Deep Reinforcement Learning

This is a framework that models each bus as an agent and develops dynamic and flexible holding control strategies for a bus route. Wang et. al. used this framework to coordinate the interactions among agents and improve learning performance [Wang, 2020]. They define a Markov decision process (MDP) and iteratively solve it with proximal policy optimization. Their MDP states and rewards depend on both passenger wait times and bus headways.

2.3 Simulation

Another approach is to determine bus holding action through simulation, given the many-dimensional nature of the problem [Adamski, 1998]. This approach is especially useful in making decisions for an entire system of buses as opposed to making an isolated decision for a few buses on a single line.

*Civil and Environmental Engineering, University of California, Berkeley
†College of Letters and Sciences, University of California, Berkeley

3 Data

In this paper, we focus on 2021 data from the Massachusetts Bay Transportation Authority (MBTA) Route 111 southbound, which takes commuters from the northern neighborhoods of Boston into downtown [MBTA, 2023]. Route 111 is one of the busiest, high-frequency, and traffic-plagued routes on the MBTA network [Vaccaro, 2018]. As such, it is a good candidate for our analysis.

Route 111 is a headway-based bus, meaning that it promises riders to arrive every m minutes as opposed to running on a fixed schedule. m is between 3 and 15 minutes depending on the time of day. Consider a sequence of buses $\{b_1, b_2, ...b_k\}$ that all pass a specific point on the road at times $\{t_1, t_2, ...t_k\}$ respectively. The headway of bus b_n at this timepoint is given by $t_n - t_{n-1}$. Route 111 has 25 stops, and our dataset contains timestamp data for 10 of them. As an example, Figure 1 demonstrates the headway evolution of one specific bus across all 10 timepoints. Although the bus is supposed to maintain 5 minutes (300 seconds) of spacing throughout its route, it bunches up with the bus ahead of it during timepoints 2-7 before increasing its headway. This is a common pattern in our data. At the beginning of the route, no bus bunching is occurring since there has been no time for a schedule deviation to occur between the bus of interest and the bus ahead of it on the route. However, as time progresses, the preceding bus may get slowed down leading to traffic, and the ensuing bus bunches.

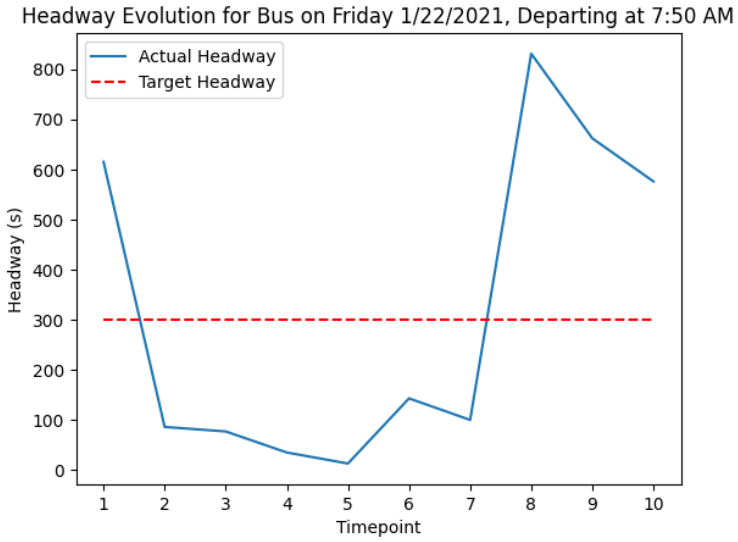

Figure 1: Example headway evolution

In our exploratory data analysis, we found that headways varied significantly and that there was a significant amount of bus bunching on the MBTA 111 route. Figure 2 visualizes the distribution of all headways in which the target headway was 5 minutes.

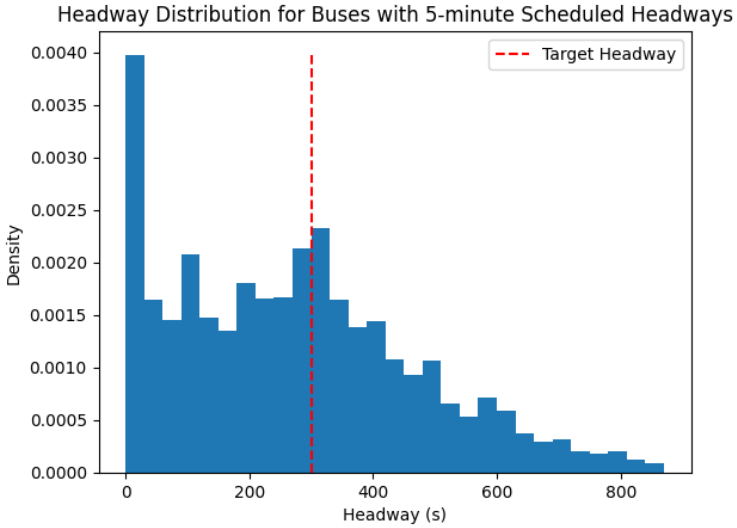

Figure 2: Headway Distribution for Buses with 5-minute Scheduled Headways

4 Problem Statement and Methodology

4.1 Machine Learning

We explored many neural network designs for time series prediction. After experimenting with different architectures including FFNN, RNN, and LSTM, we settled on FFNN as it yielded the best results.

To evaluate our models, we chose to use the Huber Loss function, given by

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y}) & \text{if } |y - \hat{y}| \leq \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases}$$

The Huber Loss function is quadratic for small errors, but linear for large errors, diminishing the effect of large errors on the overall loss function and making it less sensitive to outliers and better able to handle data with extreme values or spikes. As seen in Figure 1, our headway time series are sometimes subject to spikes in the data, making Huber Loss a good choice.

4.2 Feature Selection

In our main results, we compare two different feature sets.

4.2.1 Feature Set 1

Network architecture 1 predicts bus headways from the 3 previous headways along that route and the scheduled headway. Intuitively, this makes sense because a bus that is already bunched up will tend to stay bunched up without a control scheme.

Since our dataset provides headway information for 10 different timepoints, we seek to dynamically predict these headways using the previous 3 timesteps. Note that this means we make a prediction starting at the 4th timestep. This is reasonable because at the beginning of the route, traffic and passenger patterns typically have not built up enough to cause a deviation from the headway.

4.2.2 Feature Set 2

Network architecture 2 predicts bus headways from the 3 previous headway data points collected at that stop and the scheduled headway. The rationale behind this design is that if a certain part of the route has bad traffic, then buses will tend to bunch in a similar pattern over time. Note that this means we do not generate predictions for the first 3 buses of the day. This is largely inconsequential since the buses in the early morning hours did not experience much deviation from expected headway as there is not much traffic at that time.

4.2.3 Feed-Forward Neural Network Design

We used 6 dense layers with linear and ReLU activation functions. 50 epochs were used to generate the outputs. Our full hidden layer architecture is detailed in Table 1.

Layer	# Nodes	Activation
1	64	ReLU
2	32	ReLU
3	25	Linear
4	20	ReLU
5	10	ReLU
6	1	Linear

Table 1: Feed-Forward Neural Network Architecture

4.3 Controls

For the controls portion of the project, we fully defined a Markov decision process and used dynamic programming to find the optimal policy. On MBTA's route 111, approximately 10 buses are on the line at once. In our control scheme, we consider a line of 3 buses numbered $0, 1$, and 2. Bus 0 travels as fast as it can given current traffic conditions. Bus 1 and bus 2 may be dynamically held for some time in order to maintain proper spacing. Our controller determines the holding patterns for bus 1 and 2. Our implementation focuses on controlling buses with a target headway of 5 minutes.

4.3.1 State Space

Each element of our state space $\mathcal{S}$ is a 2-tuple of the headway of bus 1 and bus 2, respectively. We discretized the state space into 20-second intervals and only considered headways of 600 seconds or less. For our application, we do not care much about large outlier headways, since by the time the bus would be able to catch up to the preceding bus, the route would have been completed. Thus, we have $\mathcal{S} = \{0 - 19, 20 - 39, ...580 - 600\}^2$.

4.3.2 Action Space

Each timestep in our model is 2 minutes. At each 2-minute interval, there is no guarantee that the bus will be waiting at a bus stop. Therefore, we only prescribe actions that will be achievable for the bus to take in the next 2 minutes. That is, we cannot ask a bus to hold for the next 2 minutes if it is not already at a stop. Therefore, the maximum amount of time we ask a bus to hold for in a 2-minute interval is 1 minute, with the assumption that the bus will continue to the next station and begin the holding pattern there. Each action $a \in \mathcal{A}$ represents the number of seconds the buses will hold, respectively, in the next 2 minutes. Thus, we have $\mathcal{A} = \{0, 10, 20, 30, 40, 50, 60\}^2$.

4.3.3 Transition Probabilities

MBTA route 111 does not currently implement dynamic holding. Therefore, we estimated $P[s \to s' | A = 0, S = s]$ from our data simply be counting the proportion of state transitions to s' from each state s. We visualize the transition probability matrix for $a = 0$ in Figure 3. Note that the largest values are along the diagonal. This suggests that most of the time, headways do not change from one timestep to another. The further away from the diagonal, the starker the change in headway from one timestep to another. In the lower left part of the matrix, where the next headway is smaller than the current headway are states that involve bus bunching.

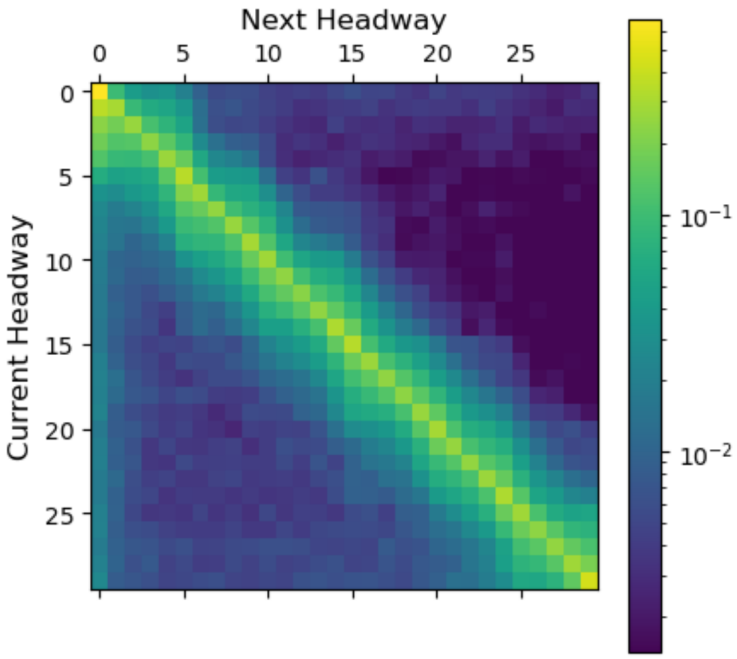

Figure 3: Transition probability matrix for $a = 0$.

For actions that involved hold time, we estimate $P[s \to s'|A = a, S = s] = P[s + a \to s'|A = 0, S = s]$. We present the probability transition matrix for $a = 60$ below. Note how some next states are infeasible since a headway cannot get smaller that the time than it is asked to hold.

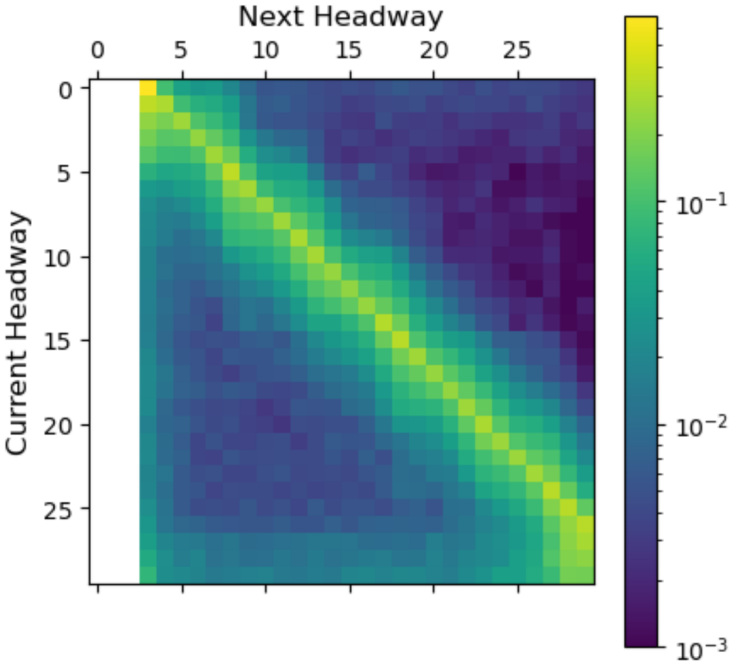

Figure 4: Transition probability matrix for $a = 60$.

4.3.4 Reward Function

Let x be our target headway. We employ the reward function $R(s) = -(s[1] - x)^2 - (s[2] - x)^2$, which penalizes both buses equally for deviations from their the headways.

4.4 Dynamic Programming

Given our full knowledge of the Markov decision process, we solved for the optimal value with dynamic programming. First, consider the dynamic programming update rule presented in class:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma P_{ss'}^a v_k(s') \right).$$

Note that in our case, the reward also depends on the next state s'. Therefore, we augment the update rule:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\sum_{s' \in \mathcal{S}} P_{ss'}^a (R_{ss'}^a + \gamma v_k(s')) \right).$$

We used $\gamma = 0.9$.

4.5 Q-Learning

Due to the computational difficulty of the problem, dynamic programming did not completely converge to the optimal policy even after running our code for many hours. Therefore, we also implemented Q-learning with SARSAMAX. For each timestep of each episode, we updated Q with the following rule:

$$Q(s,a) \to Q(s,a) + \alpha \left(R + \gamma \max_{a' \in A} Q(s,a') - Q(s,a) \right)$$

We used $\gamma = 0.9$, $\alpha = 0.01$, and $\epsilon = 0.1$.

5 Main Results and Discussion

5.1 Machine Learning

We summarize the results of our modeling in Table 2.

Feature Set	Final Huber Validation Loss
1	64.6978
2	132.0627

Table 2: Neural Network Validation Losses

Figures 5 and 6 visualize some of our predictions for feature set 1, in which we use scheduled headway and the previous 3 headways of a bus to predict its next headway. Note that the first prediction is at timestep 4, since the first 3 timesteps are used as inputs to the model.

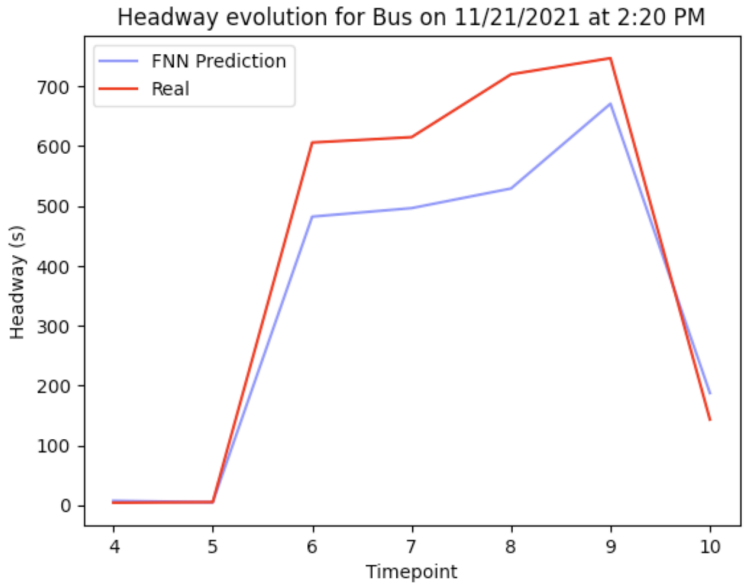

Figure 5: Headway evolution for bus departing on 11/21/2021 at 2:20 PM.

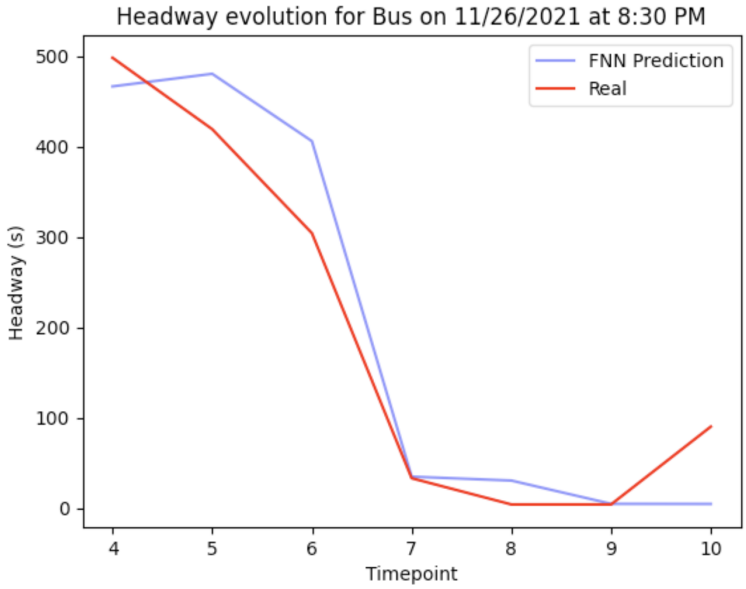

Figure 6: Headway evolution for bus departing on 11/26/2021 at 8:30 PM.

5.2 Controls

In this section we compare results from dynamic programming and Q-learning. Recall that these results correspond to controlling buses with a planned 5-minute headway. It is important to note that due to the computational difficulty of the problem, we were unable to get our policy to converge completely using either method even after running our code for many hours. Thus, our policy is slightly noisy, but still has explainable and consistent overall trends. We find that our policy derived dynamic programming was better than that of Q-learning.

5.2.1 Dynamic Programming

Figure 7 represents the value function obtained from dynamic programming. As expected, the maximum involved both headways meeting the target headway of 300 seconds.

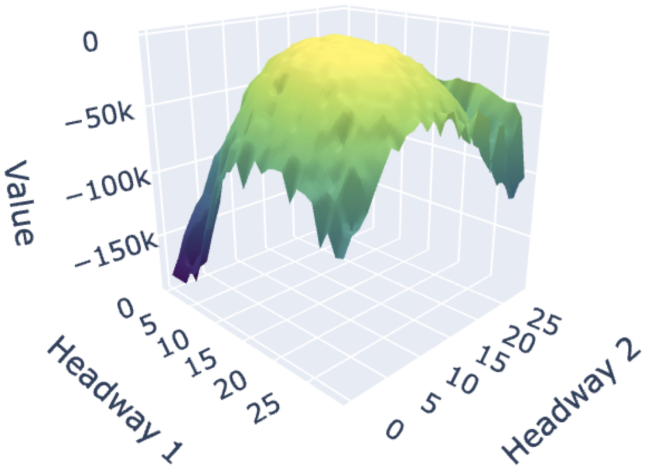

Figure 7: Value function derived from dynamic programming

Figure 8 and 10 shows the improvement in reward that our policy derived from dynamic programming obtains over the baseline policy of never holding the bus. Note that for the states in the middle of the transition matrix, where headways are already at the target, our policy does not improve over the baseline. However, in cases where the headways are unequal, we have significant improvement.

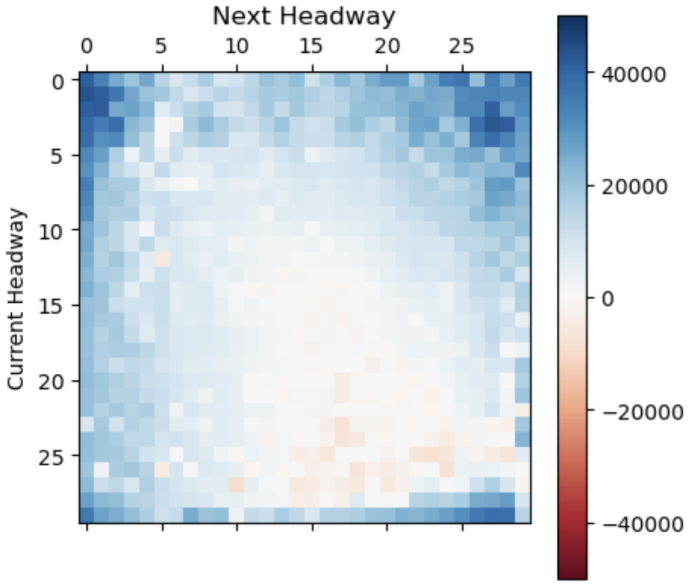

Figure 8: Difference of rewards following the dynamic programming policy and without following the policy

In Figure 9 we visualize the policy for bus 1 derived from dynamic programming. As previously mentioned, we were unable to get dynamic programming to converge completely even after running our code for many hours. However, the broad patterns are consistent. As expected, in the upper right corner of the graph, headway 1 is much larger than headway 2. Therefore, bus 1 should hold for 0 seconds and drive to catch up with bus 0. Also, in the bottom right corner of the graph, headway 2 is much larger than headway 2. Here, bus 1 should hold and give bus 2 time to catch up and re-establish proper spacing.

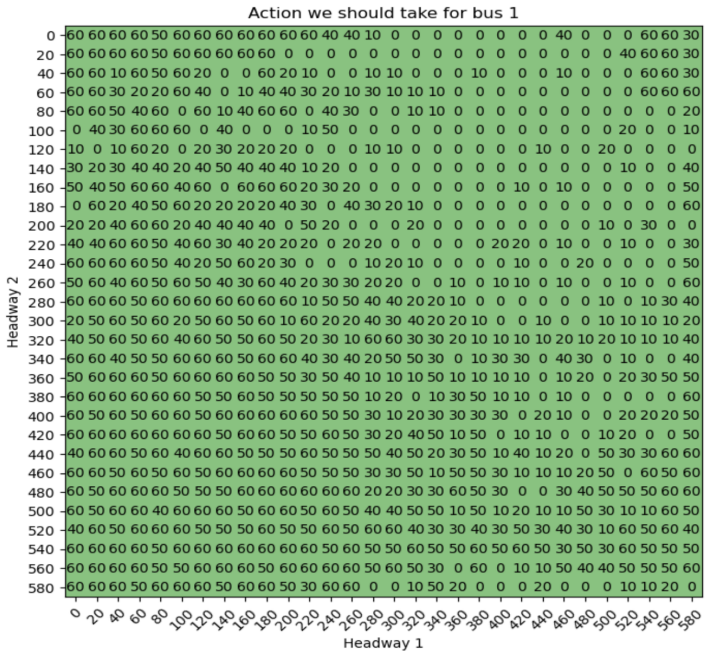

Figure 9: Bus 1 policy

5.2.2 Q-Learning

Even after tuning the learning rate and ϵ, Q-learning did not yield good results after running the code for many hours. Figure 8 and 10 shows the improvement in reward that our policy derived from Q-learning obtains over the baseline policy of never holding the bus. This policy was not as robust, and even resulted in the fact that controlling some states meant having a poorer outcome than simply doing nothing.

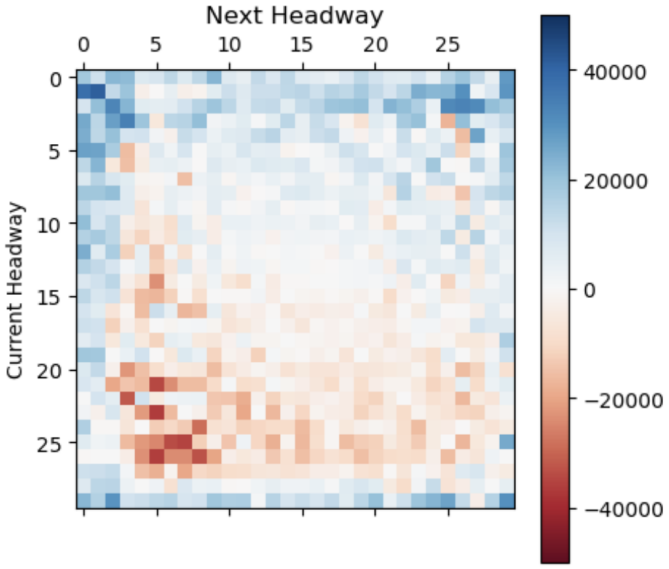

Figure 10: Difference of rewards following the Q-learning policy and without following the policy

5.3 Integration

In this section we propose an integration strategy for our machine learning and controls sections.

As the bus driver is driving, they will need some advance warning as to what action to take at the next stop. Based on the current headway patterns, the bus network's central command center predicts what the headway will be at the next stop. This prediction can be fed into a lookup table where the projected headway will determine the action and tell the driver what to do.

Note that to truly test our implementation, we would need to implement this scheme in the real world. This is because our current neural network implementation has only learned from real bus headway data *in which no dynamic holding was taken*. When simulating our policy's affect on headways, we can only predict what its headway would have been with no holding, then naively add on the effects of the holding. This approach may not hold up in reality. Also, we end up feeding predictions of the neural network back into it as inputs. However, in reality, once a driver holds the bus, they will generate *additional real data points* that would be better suited for inputs to our neural network.

However, in lieu of implementing this in the field, we assume that the predictions output by the neural network are viable enough to be fed back into the neural network to predict additional timestep. Figures 11 and 12 highlight our implementation for a pair of adjacent buses.

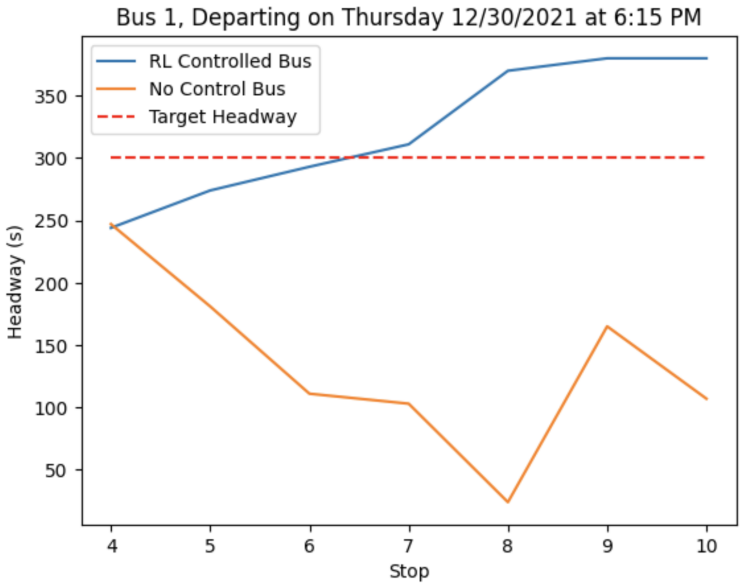

Figure 11: Bus 1 Control Implementation Example

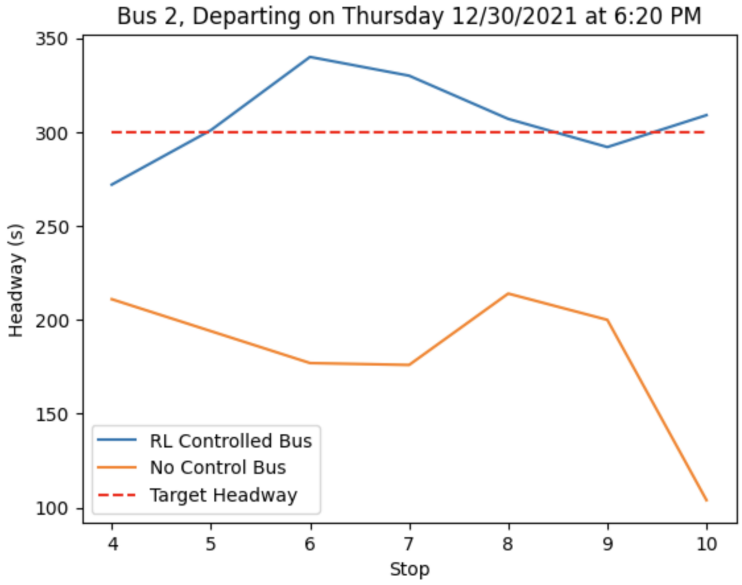

Figure 12: Bus 2 Control Implementation Example

6 Conclusion

6.1 Challenges

In retrospect, our project choice may have been too ambitious. A more holistic research project on this topic would have also considered passenger flows via smart card data, since passenger boarding and alighting is a determinant of bus timeliness. We were unable to obtain smart card data for our analysis.

Additionally, we planned to integrate traffic data from Tomtom, a paid traffic data service of which we are used the free trial [Tomtom, 2023]. We had to manually query data through Tomtom's API to obtain percentiles of different travel times for each of the segments in between our timepoints for each hour of the day and day of the week. That is, for each hour, we knew the 5th percentile travel time, the 10th percentile travel time, and so on. For our neural network, we added the travel time percentiles as input features to our neural network. We also formulated an MDP involving traffic states for our reinforcement learning problem. However, the Tomtom data did not help our results. We believe that the cause of this failure was the fact that the traffic data was not granular enough. A new MBTA 111 bus comes many times an hour, so simply knowing the travel times for the entire hour is not a powerful indicator of how buses will bunch. This manual querying and data wrangling from Tomtom's API was very time-intensive and took away our time for other parts of the project.

6.2 Future Work

In order to make our project conceptually and computationally feasible we designed a control strategy for 3 buses. However, in reality, there are approximately 10 buses on the road at once for the 111 route, depending on time of day. Given more time and computational resources, a good next step would be to expand our control scheme to accommodate an arbitrary number of buses.

Our implementation focuses on high-frequency, headway-based, urban buses. However, there are numerous other types of bus line to which this could be applied, such as schedule-based buses, shuttles that go in loops, and buses that run through non-urban areas. Thus, it would be good to implement our model on a variety of bus lines to see if it is generalizable.

Additionally, our model could likely be improved with robust data about passenger and traffic flows. Locating good data and integrating these factors into our model would also be a good next step.

References

[Adamski, 1998] Adamski, A. T. (1998). Simulation support tool for real-time dispatching control in public transport. *Transportation Research Part A: Policy and Practice*, 32.

[MBTA, 2023] MBTA (2023). *111 Bus Route*.

[Tomtom, 2023] Tomtom (2023). *Tomtom Road Traffic Analytics*.

[Vaccaro, 2018] Vaccaro (2018). *This MBTA Route Shows All the T's Bus Problems*.

[Wang, 2020] Wang, L. S. (2020). Dynamic holding control to avoid bus bunching: A multi-agent deep reinforcement learning framework. *Transportaiton Research Part C: Emerging Technologies*, 116.

[Xuan, 2011] Xuan, Juan Argote, C. F. D. (2011). Dynamic bus holding strategies for schedule reliability: Optimal linear control and performance analysis. *Transportation Research Part B: Methodological*, 45:1831–1845.