Final Project CS5180: Reducing Bus Rider Commuting Time with Reinforcement Learning

Joseph Rodriguez*, **Alexander Keane***

[1]Northeastern University
360 Huntington Ave
Boston, Massachusetts 02446

Abstract

This paper formulates an asynchronous multi-agent control task as a reinforcement learning problem to learn a centralized policy. Such problems are more commonly approached with decentralized learning and execution. The simplification is done to avoid the curse of dimensionality when considering global information during learning. However this problem may be subject to non-stationarity, given the potential of agents interfering with each others' reward signals. Moreover, it does not allow the learning of collectively optimal policies. To address this issue, this paper proposed to define a global MDP and learn on an aggregated state to speed up learning. This approach is demonstrated in the bus holding problem, in order to regularize service and reduce riders' total commuting time. Experiments are conducted using a bus simulation model as environment and tabular methods for learning. Results show that reinforcement learning methods outperform the common practice for holding.

Introduction

The primary challenge in multiagent control tasks with reinforcement learning is to define a proper goal to learn (Busoniu, Babuska, and De Schutter 2008). Methods proposed differ by the degree to which information is shared between agents (Busoniu, Babuska, and De Schutter 2006). For instance, joint learning fully shares information with a global state and action space. Although this may facilitate the learning of collectively optimal policies, it calls into question the feasibility of learning in larger-scale environments and the practicality of global observability. On the other hand, independent learning treats each agent as a single agent with its own local observations and actions. Although practical, this approach leads to environment non-stationarity from a single agent's perspective, because the single agent's future may be influenced by another agent's actions (Matignon, Laurent, and Le Fort-Piat 2012).

This paper formulates the bus holding problem, a well-studied case for multi-agent control, as a multiagent reinforcement learning problem with joint learning. To address the curse of dimensionality, state aggregation is used to significantly decrease the size of the state space and make the learning problem tractable for tabular learning methods. Experiment results are shown to compare tabular reinforcement learning methods.

*These authors contributed equally.

Related Work

This section introduces the bus holding problem and discusses previous approaches using reinforcement learning.

The bus holding problem, in which a bus can be held at a stop for longer to maintain even spacing between buses, is used as case study for multiagent reinforcement learning. For this problem, the literature consists mostly of independent learning applications. Although some papers incorporate shared rewards (Menda et al. 2019) and credit assignment (Wang and Sun 2021), several issues remain unaddressed. In these studies, the proposed rewards promote interval regularity and penalize on-board passenger wait from holding the bus. This is a missed opportunity for learning what is the central goal of transit service: get riders to destinations as quickly as possible. Another issue is that the reward does not distinguish stop location, which is problematic for bus routes with uneven demand patterns. For instance, stops that usually have more passengers going through should penalize holding more.

By reconstructing this MDP with joint observation and action space, the reward function is simplified to return the total passenger time in the system between timesteps. Such reward function captures the trade-offs from holding actions applied at scale and can lead to a more collectively effective policy. Given that a reduction in overall passenger time may take several timesteps (and appropriate actions) to take effect, such reward leverages the capability of reinforcement learning algorithms to back up long term rewards.

This paper experiments with such settings to evaluate overall travel time savings when compared to typical holding methods.

Problem Formulation

In this section, the characteristics of the bus route layout, service and the control problem are defined.

Bus Operation

Consider a bus route service serving stops in a loop, where stops 0-6 represent residential areas and stops 7-9 represent

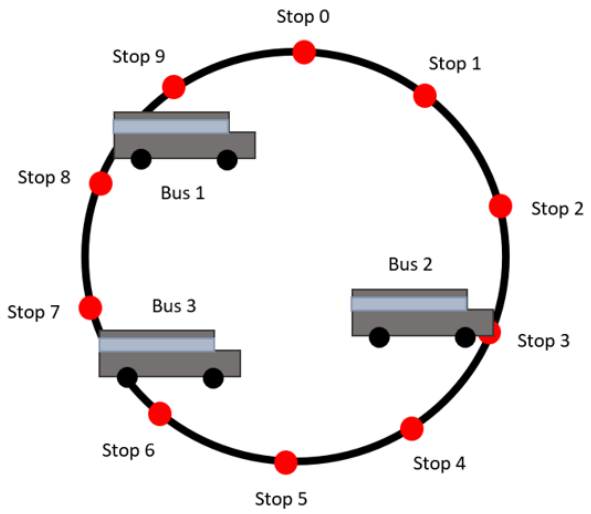

Figure 1: Model of bus operations

commercial areas. One passenger is generated at residential stops with a probability of 0.05, and passengers exit the bus at residential stops with a probability of 0.15. At commercial hubs, passengers are generated at a rate of 0.015 and passengers exit the bus with a probability of 0.5. Additionally, buses move from stop to stop in 3 minutes on average following a normal distribution. The large degree of stochasticity is meant to reproduce a realistic model to a real world example. In addition, buses have limited capacity, so passengers left behind due to lack of space must wait until the next bus.

Bus Control

The control strategy considered is of holding buses at stops if needed in order to even out the spacing between them. At any given time, the location of buses, and their fullness is fully observable. (This resembles the vehicle technology that many transit agencies use to monitor their fleet.) Also, the number of riders waiting at each stop is tracked (consider camera technology to count). The task for the centralized controller is to, based on the information available, instruct all buses whether to hold at stop or not.

Reinforcement Learning Formulation

Markov Decision Process

A Markov Decision Process (MDP) for the single-agent RL case can be formally defined as $(\mathcal{S}, \mathcal{U}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{S}$ is the set of states that can be encountered when the agent interacts with the system, $\mathcal{U}$ the set of possible control actions that can be taken, $\mathcal{R}$ the reward function based on action $u \epsilon \mathcal{U}$ taken at state $s \epsilon \mathcal{S}$ and resulting in state s'; $\mathcal{P}$ is the transition probability function which determines the probability that state s' will result, given current state s and action u (i.e. $P(s \rightarrow s'|s, u)$), and γ is the discount factor which discounts the value of future rewards compared to immediate rewards.

The MDP in this reinforcement learning problem is a public transit bus route. This consists of a state - $\mathcal{S}$, and action - $\mathcal{U}$ defined by:

$$\mathcal{S} = [PS_{0-9}, PB_{1-3}, BL_{1-3}, SL_{1-3}] \tag{1}$$

where PS - passengers at each of 10 stops (maximum of 3), PB - passengers on each of 3 buses (maximum of 4), BL - location of each of 3 buses, and SL - if each of three buses is currently at a stop or a link (between stops).

$$\mathcal{U} = [B_1, B_2, B_3] \tag{2}$$

where each bus has a binary action to go to the next stop or wait at the current stop.

Reward Function

The reward function is defined by:

$$\mathcal{R} = -\sum_{i=0}^{9} PS_n - \frac{1}{2}\sum_{i=1}^{3} PB_n \tag{3}$$

which means that each passenger waiting at a stop produces a reward of -1 every minute. This is halved for passengers on the bus. This reward is simple but directly represents the goal of the transit system. The aim is to reduce passenger travel time as much as possible, which reinforcement learning should do by providing a negative reward for passengers in the system.

Transition Probability

Defined as in the model earlier described. The outcome of the actions are dependent on the passenger demand and travel times.

Proposed Solutions

Sarsa

The temporal difference method, Sarsa, is one of the algorithms utilized to solve this reinforcement learning policy. A one step or this problem, one minute update is performed on the Q-values. This is an on policy method, meaning that the next time steps Q-value is based on the current policy. In this case an epsilon-greedy method is used. The equation for updating Q-values is:

$$Q_{t+1}(s, u) = Q_t(s, u) + \alpha \cdot [r + \gamma Q^{\pi}(s', u') - Q_t(s, u)] \tag{4}$$

where action selection is:

$$u' = \arg\max_{u} Q(s', u) * P(1 - \epsilon) \tag{5}$$

when P is 1, else:

$$u' = random(U \in u) \tag{6}$$

Since the Q-values in Sarsa are reflect the behavior policy this method is not expected to yield ideal results. Instead it is to be used as a comparison method to various other methods, such as double-Q learning, always-go policy as well as various minimum distance policies.

Algorithm 1: Sarsa

Parameter: step size $\alpha\epsilon$ (0,1], small $\varepsilon > 0$
Initialize: $Q(s, u)$ = q_0 for all s and u

for episode is not over **do**
  Initialize S.
  Choose U from S using ϵ-greedy policy
  for each step in episode **do**
    Choose U' from S' using ϵ-greedy policy
    Update $Q(S, U)$ with Equation (4)
    $S \leftarrow S'$
    $U \leftarrow U'$
  end for
end for

Algorithm 2: Q-learning

Parameter: step size $\alpha\epsilon$ (0,1], small $\varepsilon > 0$
Initialize: $Q_1(s, u)$ and $Q_2(s, u)$, for all $s\epsilon S^+$, $u\epsilon A(s)$, such that $Q(terminal,) = 0$

for each episode **do**
  Initialize S
  for each step of episode **do**
    Choose U from S using policy derived from Q (e.g., ε-greedy)
    Take action U, observe R, S'
    Update $Q(s, u)$ with Eq. (7)
    $S \leftarrow S'$
  end for
end for

Q-learning

The agents are trained with the Q-learning algorithm (Watkins 1989). Q-learning is a model-free reinforcement learning method which uses dynamic programming to iteratively update the value of an action u taken at state s, e.g. $Q(s, u)$. Upon trying all actions repeatedly, the action with highest Q-value is judged as the best action overall for that state, considering immediate rewards and discounted long-term rewards. The update is as follows:

$$Q_{t+1}(s, u) = Q_t(s, u) + \alpha \cdot [r + \gamma Q(s', u') - Q_t(s, u)] \quad (7)$$

where

$$u' = \arg\max_u Q(s', u) \quad (8)$$

and α is the learning rate. Pseudo-algorithm 2 oulines the steps.

Double Q-learning

Q-learning is shown to overestimate the state-action values, due to the maximization bias in the target value estimation. Considering the bus operation is prone to noisy reward signals, this maximization bias may lead to unstable training initially and thus require longer to learn an optimal policy. To reduce such effect, Double Q-learning (DQL) is applied (Hasselt 2010). DQL reduces the bias by maintaining 2 estimates of state-action values and choosing at random which

Algorithm 3: Double Q-learning

Parameter: step size $\alpha\epsilon$ (0,1], small $\varepsilon > 0$
Initialize: $Q_1(s, u)$ and $Q_2(s, u)$, for all $s\epsilon S^+$, $a\epsilon A(s)$, such that $Q(terminal,) = 0$

for each episode **do**
  Initialize S
  for each in-episode step **do**
    Choose U from S using the policy ε-greedy in Q_1+Q_2
    Take action U, observe R, S'
    Sample x from $X \sim uniform(0, 1)$
    if $x > 0.5$ **then**
      Update Q_1 using Eq. (9)
    else
      Update Q_2 using Eq. (9)
    end if
    $S \leftarrow S'$
  end for
end for

estimate to update at each timestep for the state-action pair. Consider estimates Q_a and Q_b. If Q_a is selected for updates (with probability 0.5), then it's estimate for the state-action pair (s, u) is updated as follows:

$$Q_a(s, u) = Q_a(s, u) + \alpha \left(R + \gamma Q_b\left(s', u'\right) - Q_a\left(s, u\right)\right) \quad (9)$$

where

$$u' = \arg\max_u Q_a\left(s', u\right) \quad (10)$$

Greedy actions are selected based on on the sum of the estimates. Psedo-algorithm 3 outlines the steps.

State Aggregation

One issue with the current state-action space is its size. The total combination of state-actions is 8.38 x 10^{12} which is much too large to learn using traditional learning methods. One option to address this problem is to use state aggregation methods. This entails consolidating the overall state action space when constructing Q-values. For example, it may not be necessary to know the exact number of passengers at every single stop to form a successful policy. One way to visualize the bus loop network is as a line (see Figure 2). In such case, only the lead bus's position is tracked. The distance between the lead bus and the second bus is aggregated to range from 0 - 2 and the same process is used for lead bus to the trailing bus. Additionally the fullness of the lead bus is tracked and aggregated.

The baseline state aggregation proposed $\mathcal{S}_o$ is as follows:

$$\begin{aligned}\mathcal{S}_o = [&LB_{0-9}, MB_{0-N_M}, \\ &TB_{0-N_T}, PL_{0-N_P}, L_{0-1}, M_{0-1}, T_{0-1}]\end{aligned} \quad (11)$$

where lead bus location (LB) is tracked and whether or not it is at a stop (s). Additionally, the middle and trailing buses aggregated locations (MB and TB) are also tracked along with if they are at a stop (M and L). The a binary value indicating the passenger level (PL) is also tracked to provide the agent more information about the state.

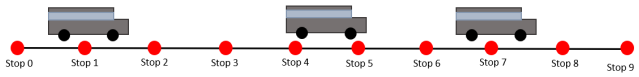

Figure 2: Bus model for state aggregation

The reasoning behind this aggregation is that it allows for minimal information to convey the position of buses. For instance, stop location is important because of the different demand patterns associated with residential and commercial hubs. Also, it is known that bus spacing plays an important role in the efficiency of the transit system so this is also tracked. Lead bus fullness intuitively appears to be a valuable data point on the action of the lead bus. It would make sense to continue moving the lead bus to the commercial hubs, to allow passengers to exit, if the bus was signaling that it is full. Finally, whether or not the bus is at a stop or not is tracked so the agent can learn whether its action has any impact on the bus movement. For example, if the bus it traveling between stops the action will have no effect on the outcome.

For more demand detail in the aggregated state, the extension $\mathcal{S}_+$ is proposed:

$$\mathcal{S}_+ = [PS_{0-3}, PA_{0-4}] \tag{12}$$

This extension includes passengers waiting at the stops (PS) and passengers ahead of lead bus but before the commercial hub (PA). The number of passengers waiting at the stops is tracked as the behavior of the agent when there are very few passengers waiting at stops should be different than when there is a full transit system. The number of passengers waiting between the lead bus and the commercial hub is tracked for similar reasons to the lead bus fullness. If there are a large number of waiting passengers between the lead bus and the first commercial hub the correct behavior would be to quickly move to service these passengers. By tracking this, the agent can learn a similar strategy.

Experiment Design

Simulation Model

A simulator is built using Python to replicate the conditions defined in the bus operation, and accommodate reinforcement learning training. Each episode initializes empty buses positioned at random location, and begins generating riders at stops. The simulation is designed to, at each step, perform the following operations: advance the clock one minute, generate new riders at stops according to statistical distributions, and process bus arrivals at stops. Afterwards, the simulation outputs the raw state and reward for the previous action taken. As is the nature of MDPs, the simulation only uses the information contained in the raw state, action and initial static inputs to determine the next state and reward.

Scenario Comparison

The performance is compared for the reinforcement learning methods, the scenario in which no holding is applied (always-go), and a scenario in which holding is applied to enforce a minimum distance between the reference bus and its leader. If the bus is less than a threshold of stops away from its leader and it is currently at a stop, then it holds until the next timestep (1 minute later). Such strategy has shown to improve service reliability and waiting times in research and real-world operations (Cats 2014), making it a realistic benchmark . The threshold distance used are 2 and 3 stops .

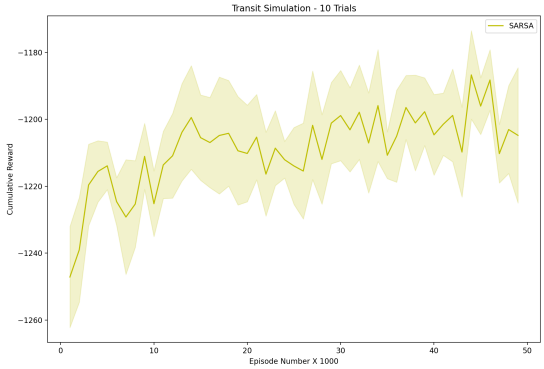

Figure 3: Sarsa algorithm cumulative rewards using over 50,000 episodes of learningon the transit environment.

Parameter settings

For each method, tuning of parameters was conducted to achieve the best performance. Table 1 lists the best parameter configurations for each method, where N represents the number of training episodes and q_0 is the initialized value of the state-action estimates.

Parameter	Sarsa	QL	DQL
α	.1	.001	.001
ε	.1	.5-.01	.6-.01
γ	.99	.99	.98
N	1000	1000	1000
q_0	-660	0	0
$\mathcal{S}$	$\mathcal{S}_o$	$\mathcal{S}_o + \mathcal{S}_+$	$\mathcal{S}_o + \mathcal{S}_+$
N_M	2	9	9
N_T	2	9	9
N_P	1	4	4

Table 1: Parameter settings for each method

Results Analysis

We first show the learning curve for SARSA with extended training. This is run for 10 trials with each trial consisting of 50,000 training episodes (see Figure 3). The majority of the learning occurs in the first 10,000 episodes and then the average rewards begins to level off. Overall, the agent learns to increase the reward from -1250 per trial to approximately -1200 per trial. This is an improvement, however the high degree of randomness in the environment make learning inefficient with this method.

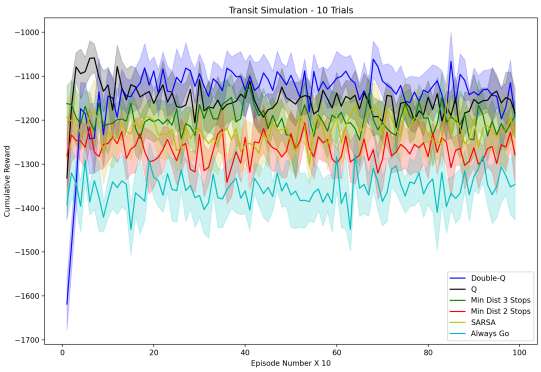

Figure 4: Double Q-learning, Q-learning, and Sarsa algorithms tested over 1,000 episodes of learning on the transit environment. Compared to control methods of minimum distance and always go.

The learning curve of scenarios are shown (see Figure 4). 10 trials of each learning and policy method are tested. When analyzing the performance of each of the algorithms it can be seen that the double Q-Learning performs the best (see Figure 4). This improvement from SARSA is surprising considering double Q-Learning uses the extended aggregated state, which creates a much larger state action space than the previous SARSA space. The most likely explanation for this finding is maximization bias can play a large role in this MDP, and double-Q learning is one of the best ways to reduce the effects of this bias. The high degree of randomness in passenger generation, which effect the reward, along with the randomness of bus travel contribute to an MDP effected by maximization bias.

The average cumulative rewards of each method is averaged for each method from episodes 200 to 1000, which is chosen because double-Q Learning no longer improve beyond this point. The best of the standard control methods is maintaining a minimum distance of 3 stops between buses. Since there are 10 bus stops, in practice this leads to even spacing between buses. The undiscounted returns of this method yield an average of -1199 (see Table 2). The best reinforcing learning method, double Q-Learning yields -1122, which is an improvement of 77. The average episode generates 72 passengers, which means that the average passenger will save more than one one minute on their morning commute when the transit system follows the double Q-learning policy as opposed to the minimum distance policy.

Conclusion

Improving the reliability of transit service has been a goal of transit agencies for a long time, especially in the case of mixed-traffic bus operations. Traditionally, policies such as maintaining a minimum distance between buses or never having the buses hold at the stops have been utilized. More recently multiagent reinforcement learning methods have utilized shared rewards (Menda et al. 2019) and credit assignment (Wang and Sun 2021) to train policy. There are potential issues with these methods as having each bus act independently may not optimize the efficiency of the system.

Parameter	Avg Reward (Epi. 200 - 1000)
SARSA	-1225
Q-Learning	-1162
Double Q-Learning	-1122
Min. Dist = 3	-1199
Min. Dist = 2	-1270
Always Go	-1354

Table 2: Average performance results

Overall, this study demonstrates a proof of concept for using a centralized agent with a high level of state aggregation to optimize bus movements in a transit system. By creating an MDP of the overall transit system, with a centralized controller, it has been shown that a more efficient policy can be formulated than traditional holding patterns. Since every bus route has its own set of unique traffic patterns and rider movement, reinforcement learning provides a very adaptable policy that can be customized to each route.

However, there is one limitation to the centralized agent method covered here. When transit routes become too large and the number of buses increases, there is the potential for dimensionality to become too large even when using the state aggregation methods employed here. A potential improvement to accomodate increased state-action spaces is policy gradient methods and deep neural networks.

References

Busoniu, L.; Babuska, R.; and De Schutter, B. 2006. Multi-Agent Reinforcement Learning: A Survey. In *Robotics and Vision 2006 9th International Conference on Control, Automation*, 1–6.

Busoniu, L.; Babuska, R.; and De Schutter, B. 2008. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2): 156–172. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews).

Cats, O. 2014. Regularity-driven bus operation: Principles, implementation and business models. *Transport Policy*, 36: 223–230.

Hasselt, H. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Matignon, L.; Laurent, G. J.; and Le Fort-Piat, N. 2012. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1): 1–31.

Menda, K.; Chen, Y.-C.; Grana, J.; Bono, J. W.; Tracey, B. D.; Kochenderfer, M. J.; and Wolpert, D. 2019. Deep Reinforcement Learning for Event-Driven Multi-Agent Decision Processes. *IEEE Transactions on Intelligent Transportation Systems*, 20(4): 1259–1268. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

Wang, J.; and Sun, L. 2021. Reducing Bus Bunching with Asynchronous Multi-Agent Reinforcement Learning. *arXiv:2105.00376 [cs]*. ArXiv: 2105.00376.

Watkins, C. 1989. Learning from delayed rewards. 1989. *University of Cambridge*.