Reinforcement learning for bus bunching problem with event-driven process and parameterized action spaces

Jung Hoon Cho
MIT
jhooncho@mit.edu

Ao Qu
MIT
qua@mit.edu

Abstract

Bus-bunching, where groups of buses arrive at one station at the same time, is a long-standing problem in public transportation operations. Having bus bunched together can hugely affect the system efficiency as it will leave a large gap in bus schedules and make passengers' wait longer. Previous works have studies how to let buses hold at stations to mitigate this problem. In this paper, we propose a new problem setting where a bus can choose from a combination of actions including holding, skipping, and turning around. We use reinforcement learning to explore the best strategy that allows multiple buses coordinate and learns the optimal strategy to improve system efficiency. To address this problem, we propose an attention-based state encoder and use Proximal Policy Optimization (PPO) with hybrid action space and temporally discounted reward factor to learn the optimal policy. Our preliminary results demonstrate that we can significantly reduce bus bunching with our proposed method.

1 Introduction

Bus bunching problem has been a long-discussed problem to avoid the service unreliability of public transportation. If there is a delay in a bus, there will be an increased number of passengers waiting at the station. This delay might increase the boarding or alighting time at the station, leading the following vehicle to stay longer. After a while, the system will encounter the bus bunching phenomena [2]. As such, several factors in the bus system, such as road traffics, drivers, or passengers, can cause variable delays and make the system unstable [6]. This unbalanced system can make the bus travel in a bunch rather than evenly distributed. This problem can cause the cost of longer travel time delays for each passenger. Many studies have tried to overcome this challenge with flexible control strategies, including bus-holding control, leap-frogging (skipping), and so on. Recently there have been attempts to apply reinforcement learning to deal with this problem since reinforcement learning has its strength in solving sequential decision-making problems. Operation of identical buses on the same route is often characterized as a decentralized multi-agent system [14, 5, 1, 13, 10].

This study aims to develop a reinforcement learning-based model to solve the bus bunching problem with an event-driven process and parameterized action spaces. In this problem, there might have different timing of the executing actions that the agent needs decisions. To handle this issue, we can formulate our decision process as an event-driven decision process [5]. On the other hand, the actions the agent takes could be divided into two levels; the high level being the choice of the control strategy, and the low level being the duration of bus holding. There are several control strategies that have been considered as possible solutions to the problem. Here we consider holding, skipping, and turning around actions. There have been attempts to treat this problem as the multi-agent version of the Markov decision process since the buses make independent and decentralized decisions. However, having the perfect observation of all buses, singe-agent problem is sufficient to solve this problem. Lastly, since the bus operating system has multiple aspects of being considered, the simulator was

Course Project of 6.7950 Reinforcement learning, Fall 2022.

designed for its own purpose to evaluate our policy. As a reward, the trained policy should minimize the variance of each passenger or the sum of waiting times (including in-vehicle and out-of-vehicle waiting times) for all passengers.

The remainder of this paper covers related works, methodologies, simulator design, experiment setups, and results. This paper ends with a discussion and offers outlooks for the future research.

2 Related work

2.1 Bus bunching problem

Daganzo proposed the strategy to insert the slack based on the headway to address the bus bunching problem [2]. With the GPS-based location data, the strategy based on the relative location of all buses and users highly contributed to reduce the bus bunching [6]. Following work proposes adjusted the bus speed based on the spacing of the consecutive buses [3]. Also dynamic holding strategy was developed by [15]. With this problem being sequential decision making, recent work focuses on the reinforcement learning based strategy to reduce bus bunching[14, 5, 1, 13, 10].

2.2 Event-driven decision process

Event-driven process refers to scenarios where the action should be taken only when an event occurs. Because the duration between two events is stochastic, the sequence and timing of actions are governed by an underlying stochastic process. [5] discussed the event-driven decision process in the context of deep reinforcement learning and used bus-holding control and fighting wildfire with unmanned aircrafts as two illustative examples. They proposed to use the temporally extended factor to discount the reward but did not give a practical way for calculation.

2.3 Parameterized action spaces

Policy gradient methods proposed in [11] recently showed relatively high performance in multiple settings. It maximizes the expected policy value by optimizing the stochastic policy π_θ, which is parameterized by θ. Also, some of them are computed with the advantage function $A^{\pi_\theta}(s,a)$ rather than the Q value function. Recently, trust region policy optimization (TRPO) and proximal policy optimization (PPO) based on these policy based methods are developed and improved overall performances [7, 9]. With PPO based on the actor-critic framework, new framework needed to handle the aggregated action spaces. The prior work [4] considered parameterized action space, which contains the selected discrete action of a and its parameterized another action x executed with the higher level action a. As a result, the action space is built on top of the tuple, which has a first component of a and a second component of x.

$$\mathcal{A} = \bigcup_{a \in \mathcal{A}_d} \{(a,x) \mid x \in \mathcal{X}_a\}$$

Since we have large number of space for the actions, we have large number of parameters for the action-value function $Q(s,a,x_a)$. When computing the variance-reduced advantage function estimator $\hat{A}$, the state value function $V(s)$ is calculated from the critic network. The policy for discrete actions π_{θ_d} and the continuous policy π_{θ_c} are to be updated by minimizing the respective clipped surrogate objective loss $L_c^{CLIP}(\theta_c)$ and $L_d^{CLIP}(\theta_d)$ [4].

$$L_c^{CLIP}(\theta_c) = \hat{\mathbb{E}}_t[\min(r_t^c(\theta_c)\hat{A}_t, clip(r_t^c(\theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$
$$L_d^{CLIP}(\theta_d) = \hat{\mathbb{E}}_t[\min(r_t^d(\theta_d)\hat{A}_t, clip(r_t^d(\theta_d), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

Also the probability ratio for discrete policy $r_t^d(\theta_d) = \frac{\pi_{\theta_d}(a|s_t)}{\pi_{\theta_{d(old)}}(a|s_t)}$ and that for the continuous policy $r_t^c(\theta_c) = \frac{\pi_{\theta_c}(x_a|s_t)}{\pi_{\theta_{c(old)}}(x_a|s_t)}$ should be considered separately [4].

2.4 Transformer

Tranformer was first introduced in [12] and has achieved many successes in fields such as natural language processing and computer visions. The key of transformer is the use of self-attention mechanisms, which allow the model to selectively focus on different parts of the input sequence at different times, rather than processing the entire sequence in a fixed manner as is the case with recurrent networks.

3 Methods

3.1 Problem Formulation

We define the bus bunching problem as a single agent markov decision process. The environment consists of a fleet of N buses $\mathbb{B}_i$ and a set of M stations $\mathbb{S}_i$. All buses are indexed in a sequential order such that bus $\mathbb{B}_{i+1}$ is in front of $\mathbb{B}_i$. Similarly, stations are ordered along the bus route. While previous works have mostly only modelled single direction of bus operation, we consider both directions to better reflect the reality. In this way, $\mathbb{S}_1$ and $\mathbb{S}_M$ both refer to the terminal station while all buses start their trip at $\mathbb{S}_1$ and end their trip at $\mathbb{S}_M$. In addition, during regular operation, the bus will alight all passengers at station $\mathbb{S}_{M/2}$ and turn to the station $\mathbb{S}_{M/2+1}$. A visual illustration of the station layout is shown in Fig 1. Except for $\mathbb{S}_M$ and $\mathbb{S}_{M/2}$, every station will have passengers arrive throughout the simulation following a time-dependent arrival rate. While the distances between stations are pre-determined, travel times between every two consecutive stations change over time. In our study, we decide to use non-constant arrival rates and travel times to simulate the temporal variation in traffic patterns.

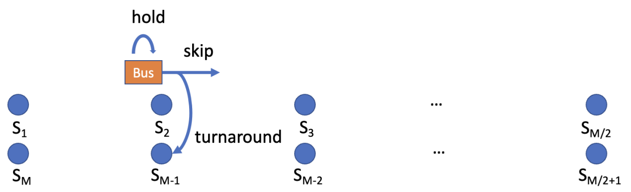


Figure 1: Station Layout and Bus Action Space

3.1.1 Observation

In our study, we consider two types of design for observation space. One is global observation which contains information about all buses. For each bus i at time t, the state $s_t^{(i)}$ contains [$\texttt{location}_t^{(i)}$, $\texttt{headway}_t^{(i)}$, $\texttt{number of passengers}_t^{(i)}$, $\texttt{high-level action}_t^{(i)}$, $\texttt{low-level action}_t^{(i)}$, $\texttt{action duration}_t^{(i)}$]. In this representation, `headway` refers to the expected travel time for this bus to get to the location of next bus and `action duration` refers to how long the action has been executed if there is an on-going action. We add information about the action each bus is taking so that the markov property can be satisfied. To represent the global state, we simply concatenate the states of all buses and shift their order such that the first bus would be the ego bus which is about to take an action. Another observation space we consider is to only use the local information which contains [`location`, `headway`, `number of passengers`] of the ego bus.

3.1.2 Action

We adopt a hierarchical action space where a bus may choose one high-level action and one low-level action. At each stop, a bus may choose one of up to three possible high-level actions which are `Holding`, `Skipping`, and `Turning Around`. `Holding` means that the bus will hold for T_H at the station after it boards and alights passengers. `Skipping` means that the next station will be skipped and this action is only allowed if no passenger on the bus needs to get off at next station. `Turning Around` is to have the bus alight all passengers and then drive to the opposite station to start serving the other direction after waiting for T_T. For `Holding` and `Turning Around`, we need a low-level action which defines T_H and T_T. Therefore, our action space is defined as $\{k, X_k\}$ where $k \in \{1, 2, 3\}$ denotes one of the three possible discrete actions and $X_k \in [0, T_{threshold}]$ denotes the continuous part which will be executed if needed.

3.1.3 Reward

The ultimate goal of all bus control strategies is to reduce the passengers' waiting time. To achieve this goal, a natural reward definition is the negative accumulative passengers' waiting time since last action. Specifically,

$$r = -(\alpha * \texttt{Total On Bus Time} + \beta * \texttt{Total Off Bus Time})$$

where we use α and β to balance the weights of passengers' time spent on bus and off bus. In practice, it often takes much more training to maximize this reward because it requires the agent to plan for longer horizons. Therefore, we also consider a proxy reward which is the negative of variance in headways between buses,

$$\hat{r} = -\frac{\sum_i^N (\texttt{headway}^{(i)} - \overline{\texttt{headway}})^2}{N}$$

3.2 Model structure

For the bus bunching problem, we identify several key challenges when designing the algorithm. First, since we are observing all the states of all agents while action is applied to only one agent. The state encoding should be order-invariant such that different buses should have the same policy if there is an isomorphism between two states $[s_t^{(1)}, \ldots, s_t^N]$ and $[s_t'^{(1)}, \ldots, s_t'^{(}N)]$. Second, our model should be able to output both a discrete action and a continuous action at the same time. Also, since X_k has dependence on k, the model should learn different policies for outputting X_i and X_j when $i \neq j$. Third, the discounting factor for reward should be temporally extended such that the discounting factor $\gamma(t_i)$ should be less than $\gamma(t_j)$ if $t_i > t_j$ where t_i, t_j denote the amount of time between last event and current event. In order to address these three challenges, we use a self-attention based state encoder, an actor-critic based network with hybrid action space as our policy network, and a temporally extended discounting factor.

3.2.1 State Encoding

In order to leverage global information to achieve the optimal results, we use self-attention mechanism to encode the information of all buses. For a stack of n transformer encoder with k heads and d_{model} dimensional keys, values, and queries, we will take the output of the last multi-head attention layer corresponding to ego bus which would be

$$Enc(s) = \texttt{Concat}(Z_1, \ldots, Z_k)$$

$$Z_i = \texttt{softmax}\left(\frac{Q_{n-1}^i[\texttt{ego}](K_{n-1}^i)^T}{\sqrt{d_{model}/k}}\right)V_{n-1}^i$$

$\texttt{ego}$ is the index for ego bus so that the information is obtained regarding the ego bus and order-invariance is achieved.

3.2.2 Hybrid PPO

To output a hybrid action space, we adapt the model proposed in [4]. Instead of having separate state encoders in actor and critic, we let them share the same attention-based state encoder so that the parameter spaces of actor and critic become more compatible. Then, we use two MLPs for actor and critic to output the action and estimated value. A visual illustration of the adapted H-PPO can be found in Appendix.

3.2.3 Temporally Extended Discounting Factor

Instead of using a fixed γ as the discounting factor, we use $\gamma(t) = \exp(-t\gamma)$. In this way, the Q value of a state-action pair can be estimated in the following way,

$$Q(s_1, a_1) = r_1 + e^{-\gamma t_1} V(s_2)$$

Following the definition given in [8], we also derive the calculation for generalized advantage estimation with this modified discounting factor.

$$\delta_k = r_k + e^{-\gamma t_k} V(s_{k+1}) - V(s_k)$$

$$\hat{A}_k = \delta_k + \lambda e^{-\gamma t_k} \hat{A}_{k+1}$$

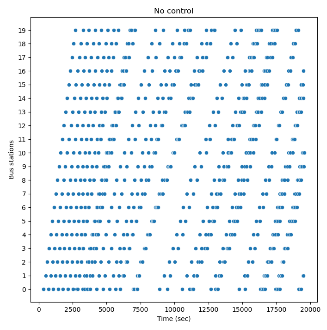
3.3 Simulation setup

We build our event-driven simulator using SimPy, a process-based discrete-event simulation framework based on standard Python. For numerical experiments, we use 12 buses and 20 stations with stochastic travel times and passengers' arrival rates. The simulator is available on https://github.com/quao627/RLBus and all parameters can be easily changed.

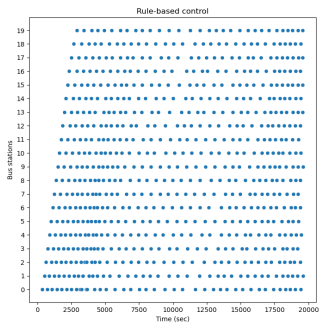
4 Results

4.1 Rule-based control

We developed a simple but efficient rule-based control for our baseline model. When the bus approaches the station, the goal is to hold their action for a certain amount of time to make headways remain similar. For example, if the ego bus's and the following vehicle's headways are a and b, respectively, our rule-based holding control time is $\frac{|a-b|}{2}$. By using this strategy, we can improve our results significantly. This simple controller assumes not having the passenger boarding and alighting time at the station. The time-space diagram plots the position of each vehicle with its location and time steps. The bus bunching phenomenon occurs if the bus operating system is not controlled (Fig. 2(a)). However, our rule-based strategy helps eliminate the bus bunching problem and shows more evenly distributed bus operation (Fig. 2(b)). In other words, the rule-based strategy helps decrease the variance of the headways of all buses. The even headways of operating buses help reduce the likelihood of bus bunching. However, the headway for the stations on the middle of the routes turned out to be larger because of their higher arrival and alighting rates.



(a) Uncontrolled situation



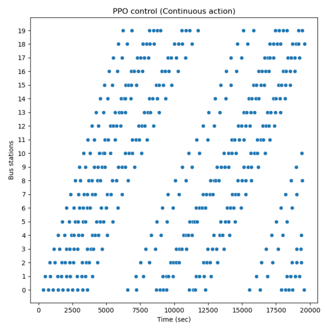
(b) Rule-based control

Figure 2: Time-space diagram for uncontrolled situation and rule-based control

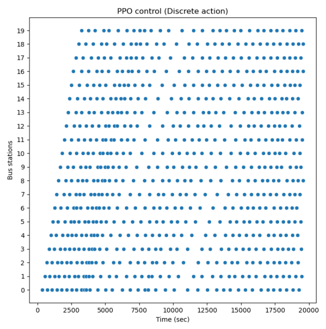
4.2 HybridPPO with MLP

Discretized action space Before using HybridPPO with complex neural networks, we can just start with the simple neural network layer to be included in our actor and critic networks to train the single bus holding strategy. However, it was hard to get good results dealing with the bus bunching in the continuous action space for the low-level actor network. The controller with continuous action spaces shows even worse bus bunching results (3(a)). Since the continuous action space is difficult to learn, we conducted an experiment to discretize the low-level action space and then train with it. The time-space diagram depicted in 3(b) shows that discretized action space is good at handling bus-bunching problem.

HybridPPO The model chooses the most appropriate action from the discrete actor among the three possible actions, which are holding, skipping, and turning around. 4(a) shows the training curve for the algorithm. Two different colors denote different runs. The time-space diagram depicting the bus locations demonstrates that considering multiple high-level actions tends to make the bus operate in a complex way (4(b)). For instance, if the bus executes the turning around action, it should alight all the passengers at the stop and proceed to the opposite direction, and wait for a while, as decided



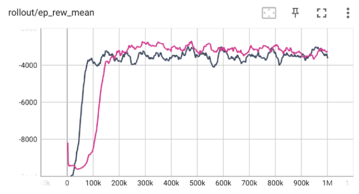
(a) Continuous low-level action space



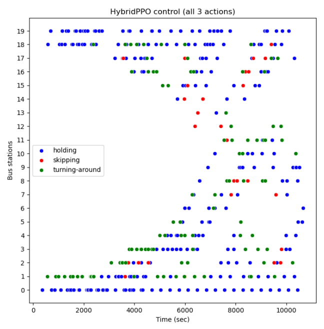
(b) Discretized high-level action space

Figure 3: Continuous and discretized action space for lower level action

by the low-level model. Is can benefit the reward of reducing the headway, but less realistic when applied to the real world.



(a) Training curve



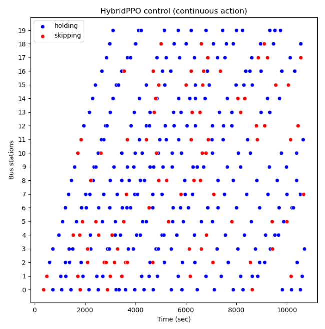
(b) Timespace diagram

Figure 4: Hybrid PPO control with all actions

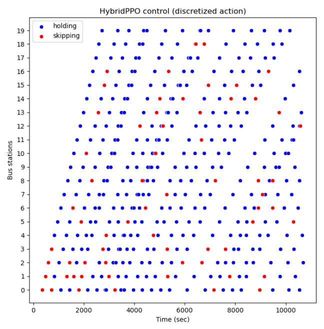
HybridPPO without turn-around control With the turning around action being less realistic to be applied to the real-world problem, we trained the model without the turning around control. Now we consider HybridPPO having two high-level actions, holding or skipping. If you compare the results from 5, still the continuous low-level action space makes the problem difficult to solve. The discretized low-level aciton space could contribute to reduce bus bunching as well as coordinate between two different high-level action. If the bus with a few passengers needs to skip the station, it makes the decision to skip the station to avoid bus bunching happens. Still this leaves us to decide the proper shape of low-level action space.

5 Discussion

Throughout this paper, we proposed policy-based reinforcement learning algorithm with the event-driven process and parameterized aciton spaces to reduce the bus bunching problem. With reinforcement learning having strength in sequential decision making, we could have outperforming preliminary results reducing the bus bunching problem in some settings. HybridPPO algorithm considering the parameterized action spaces can contribute to solve bilevel decision making problem. But still designing proper shape of observations, actions or rewards matters significantly. Reducing this phenomena is quite essential to make the public transit system more efficient, stable, and sustainable. This paper has some limitation that we have not been able to achieve very good results when using continuous action space. Also, it takes the agent longer to learn, if the policy converges at all,



(a) Continuous low-level action space



(b) Discrete low-level action space

Figure 5: Hybrid PPO

when we use global state as the input or passengers' cumulative waiting time as the reward. If we leverage the real bus-operating and location-based data without the hard assumptions made withing the simulator, we can achieve more results similar to the real-world. In the future, we will work on how to more efficiently encode the information most related to the ego bus while trying to capture the markov property. We will also try to develop reward function that aligns better with the goal while allowing the agent to effectively learn. In addition to methodology, we also plan to build simulators based on real world data to show our model's applicability.

References

[1] Weiya Chen, Kunlin Zhou, and Chunxiao Chen. Real-time bus holding control on a transit corridor based on multi-agent reinforcement learning. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 100–106, Rio de Janeiro, Brazil, November 2016. IEEE.

[2] Carlos F. Daganzo. A headway-based approach to eliminate bus bunching: Systematic analysis and comparisons. *Transportation Research Part B: Methodological*, 43(10):913–921, December 2009.

[3] Carlos F. Daganzo and Josh Pilachowski. Reducing bunching with bus-to-bus cooperation. *Transportation Research Part B: Methodological*, 45(1):267–277, January 2011.

[4] Zhou Fan, Rui Su, Weinan Zhang, and Yong Yu. Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space, May 2019. arXiv:1903.01344 [cs, stat].

[5] Kunal Menda, Yi-Chun Chen, Justin Grana, James W. Bono, Brendan D. Tracey, Mykel J. Kochenderfer, and David Wolpert. Deep Reinforcement Learning for Event-Driven Multi-Agent Decision Processes. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1259–1268, April 2019. arXiv:1709.06656 [cs].

[6] Joshua Michael Pilachowski. *An Approach to Reducing Bus Bunching*. PhD thesis, UC Berkeley: University of California Transportation Center., December 2009.

[7] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, April 2017. arXiv:1502.05477 [cs].

[8] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. arXiv:1707.06347 [cs].

[10] Haotian Shi, Qinghui Nie, Sicheng Fu, Xin Wang, Yang Zhou, and Bin Ran. A distributed deep reinforcement learning–based integrated dynamic bus control system in a connected environment. *Computer-Aided Civil and Infrastructure Engineering*, 37(15):2016–2032, December 2022.

[11] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. arXiv:1706.03762 [cs].

[13] Luca Vismara, Lock Yue Chew, and Vee-Liem Saw. Optimal assignment of buses to bus stops in a loop by reinforcement learning. *Physica A: Statistical Mechanics and its Applications*, 583:126268, December 2021.

[14] Jiawei Wang and Lijun Sun. Dynamic holding control to avoid bus bunching: A multi-agent deep reinforcement learning framework. *Transportation Research Part C: Emerging Technologies*, 116:102661, July 2020.

[15] Yiguang Xuan, Juan Argote, and Carlos F. Daganzo. Dynamic bus holding strategies for schedule reliability: Optimal linear control and performance analysis. *Transportation Research Part B: Methodological*, 45(10):1831–1845, December 2011.