Transportation Research Part C 129 (2021) 103239

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Optimizing matching time intervals for ride-hailing services using reinforcement learning

Guoyang Qin [a], Qi Luo [b], Yafeng Yin [c,*], Jian Sun [a], Jieping Ye [d]

[a] Key Laboratory of Road and Traffic Engineering of the State Ministry of Education, Tongji University, Shanghai 201804, China
[b] Department of Industrial Engineering, Clemson University, Clemson, SC 29634, United States
[c] Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI 48109, United States
[d] DiDi AI Labs, Didi Chuxing, Beijing 100085, China

ARTICLE INFO

Keywords:
Ride-hailing service
Online matching
Reinforcement learning
Policy gradient method

ABSTRACT

Matching trip requests and available drivers efficiently is considered a central operational problem for ride-hailing services. A widely adopted matching strategy is to accumulate a batch of potential passenger-driver matches and solve bipartite matching problems repeatedly. The efficiency of matching can be improved substantially if the matching is delayed by adaptively adjusting the matching time interval. The optimal delayed matching is subject to the trade-off between the delay penalty and the reduced wait cost and is dependent on the system's supply and demand states. Searching for the optimal delayed matching policy is challenging, as the current policy is compounded with past actions. To this end, we tailor a family of reinforcement learning-based methods to overcome the curse of dimensionality and sparse reward issues. In addition, this work provides a solution to spatial partitioning balance between the state representation error and the optimality gap of asynchronous matching. Lastly, we examine the proposed methods with real-world taxi trajectory data and garner managerial insights into the general delayed matching policies. The focus of this work is single-ride service due to limited access to shared ride data, while the general framework can be extended to the setting with a ride-pooling component.

1. Introduction

Assigning trip requests to available vehicles is one of the most fundamental operational problems for ride-hailing platforms, who act as a mediator that sequentially matches supply (available drivers) and demand (pending trip requests) (Alonso-Mora et al., 2017; Wang and Yang, 2019). The common objective is to maximize the system revenue or minimize passengers' average wait time, both of which necessitate devising efficient matching strategies (Zha et al., 2016).

A matching strategy widely adopted by ride-hailing platforms is to formulate and solve a bipartite matching problem between available vehicles and trip requests (Xu et al., 2018). As illustrated in Fig. 1, intentionally controlling the matching time interval to delay the matching of passengers and drivers can adjust the buffer size of potential matches and improve the operational efficiency, because drivers may find a *better* (i.e., closer, more profitable, or more compatible) match when demand accumulates and vice versa. As a result, an obvious trade-off arises when extending this matching time interval. On the one hand, the outcome of matching is

* Corresponding author.
E-mail address: yafeng@umich.edu (Y. Yin).

https://doi.org/10.1016/j.trc.2021.103239
Received 24 November 2020; Received in revised form 27 May 2021; Accepted 27 May 2021
Available online 17 June 2021

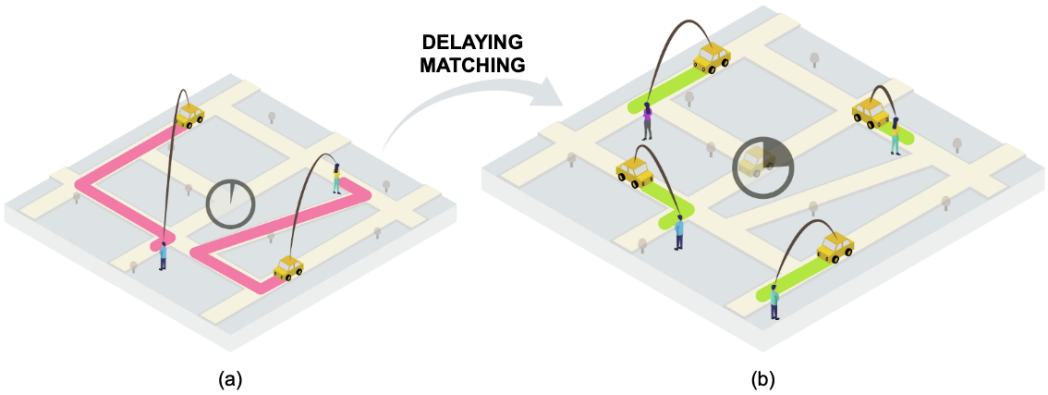

Fig. 1. An appropriate matching delay can increase the passenger-driver buffer size and improve the operational efficiency. (a) Passengers and drivers may suffer rather long pickup wait times under an instantaneous matching scheme, and (b) moderately delaying the matching can significantly reduce the pickup wait times.

improved with the increasing length of passengers' and drivers' queues; on the other hand, waiting induces *penalty* costs for each driver and passenger in queues and, in worst-case scenarios, leads to abandonment. This type of rewarding delay effect is prevalent in online matching markets. Various online platforms, such as dating applications or organ donation systems, have adopted this kind of delayed matching strategy (Azar et al., 2017).

Upon closer inspection into the dynamic matching process, one may see that the appropriate matching time interval depends on the ride-hailing system's *state*, i.e., spatiotemporal distributions of supply and demand. One of this work's main findings is that, when either the supply or demand is scarce, *instantaneous matching* should be used to maximize the system throughput. However, when the supply and demand are relatively balanced, strategically controlling the matching time interval is superior. There has been a significant research interest in the optimal control policy for the latter case (Azar et al., 2017; Ashlagi et al., 2018; Yang et al., 2020). Limitations in prior studies include requiring parametric models to obtain analytical solutions or mandating unrealistic assumptions on the system's stationarity. Nevertheless, these studies' numerical experiments show that the optimal matching time interval is distinctly sensitive to these assumptions. Hence, these model-based approaches are not implementable in complex ride-hailing applications.

Due to the state space's large dimensionality, including vehicle locations, demand forecast, and passenger characteristics, obtaining a pragmatic and effective *nonparametric* control policy for the matching time interval remains an open question. Moreover, when the system uses a fixed matching time interval, there is no prior information on the ride-hailing system dynamics under *alternative* time intervals. Reinforcement learning (RL) is therefore an obvious choice to handle this task. This work aims to develop an efficient, data-driven approach to find the optimal matching time intervals. The focus of this work is single-ride service due to limited access to shared ride data, while the general framework can be extended to the setting with a ride-pooling component.

The main contributions of this work include:

a. Proposing a dedicated reward signal for passenger-driver matching with delay, which addresses the sparse reward issue related to a long matching time interval.
b. Providing a remedy to the spatial partitioning trade-off between the state representation error and the optimality gap of local matching. This trade-off arises when we partition the area and apply asynchronous policies over those grids.
c. Presenting managerial insights for an optimized matching delay based on experiments with real-world data. The RL algorithm identifies the optimized delayed matching policy delay under medium driver occupancy.

The remainder of this paper is organized as follows. We first review the related literature in Section 2, then provide a concrete formulation of the sequential decision problem in Section 3. We investigate three RL algorithms in Section 4 as each one is a building block for the more advanced one. Finally, we show the results in a real-world case study in Section 5 and draw the final conclusion in Section 6.

2. Literature review

Online matching strategies are characterized by two spatiotemporal variables in general: the matching time interval and the matching radius (Yang et al., 2020). The platforms dynamically adjust these variables to improve the system's performance measures such as the gross revenue or cumulative system throughput. The effectiveness of matching strategies is critical for the platform's profitability and the quality of service perception of customers (Wang and Yang, 2019). Moreover, due to the negative externalities induced by empty-car cruising (Luo et al., 2019), developing advanced matching strategies ensures the sustainability of ride-hailing services.

A number of studies in the literature obtained analytical results for optimal matching strategies assuming that the system dynamics is known. Most matching strategies developed in the past consider instantaneous matching, i.e., the platform assigns trip requests to drivers upon arrival. For example, considering instantaneous matching, Xu et al. (2020b) investigated the impact of matching radius

on the efficiency of a ride-hailing system and theoretically proved that adaptively adjusting matching radius can avoid throughput loss due to "wild goose chases." Aouad and Saritaç (2020) modeled the dynamic matching problem as a Markov decision process with Poisson arrivals and developed a 3-approximation algorithm for the cost-minimization with uniform abandonment rates. They also showed that the batched dispatching policy's performance can be arbitrarily bad in the cost-minimization setting, even with the optimally tuned batching intervals. Özkan and Ward (2020) studied the optimal dynamic matching policy in a ridesharing market modeled as a queueing network. Since solving the exact dynamic program for the joint pricing and matching problem with endogenous supply and demand is intractable, they proposed a continuous linear program-based forward-looking policies to obtain the upper bounds of the original optimization. The value of delayed matching is obvious for two-sided online service systems (Azar et al., 2017). As the thickness of the market increases over time, passengers and drivers are provided with superior candidates for potential matching with a cost of delay. Ashlagi et al. (2018) studied the near-optimal matching strategies with either random or adversarial arrivals of passengers and drivers. The approximation ratio for the random case is constant, which means the algorithm's efficiency is unchanged for large networks. However, challenges remain as the deadlines for matching were given in their work. Yang et al. (2020) separated the optimization of the matching radius and matching time interval intending to maximize the net benefit per unit time. They obtained the optimal matching time interval only under the excess supply scenario and ignoring passengers' abandonment behaviors. Yan et al. (2020) studied the jointed matching and pricing problem in a batched matching setting. The wait time and matching time trade-off was identified in the equilibrium analysis. In addition, high carry-over demand or supply impeded their analysis. Most of these analytical results focused on the non-ridepooling setting.

Practitioners are more concerned about developing purely data-driven approaches for passenger-driver matching in ride-hailing services. Since online matching is a sequential decision under uncertainty, reinforcement learning (RL) is a powerful technique to adaptively improve the matching policy (Li et al., 2019; Al-Abbasi et al., 2019; Lin et al., 2018; Ke et al., 2020; Mao et al., 2020). The main technical challenge is the curse of dimensionality due to the large state space – the system contains a large number of cruising drivers and waiting passengers. The benefit of matching a specific passenger-driver pair, i.e., the reward function, is determined by the geographical information and characteristics. Li et al. (2019) approximated the average behavior of drivers by mean field theory. Al-Abbasi et al. (2019) used a distributed algorithm to accelerate the search of near-optimal vehicle dispatching policies. Mao et al. (2020) studied the vehicle dispatching policy using an Actor-Critic method. These studies require instantaneous matching to avoid many of the difficult reward design problems. Ke et al. (2020) adopted an RL-based method for the delayed passenger-driver matching problem. They employed a two-stage framework that incorporated a multi-agent RL method at the first stage and a bipartite matching at the second stage. Their work addressed several technical issues that were unresolved in previous studies, such as learning with delayed reward and value error reduction. By combining a queueing-type model with Actor-Critic with experience replay (ACER), we improve the practicability and stability of RL approaches for passenger-driver matching with delay.

3. Problem description and preliminaries

This section first formulates the delayed matching problem as a Markov decision process (MDP). We then show the existence of optimal matching time interval due to the delayed matching trade-off. Lastly, we describe how to create a simulator for agents to learn optimal policies with streaming information from a ride-hailing system, which is of particular interest to the transportation field.

3.1. Delayed matching problem in ride-hailing

In the process of online passenger-driver matchmaking, a matchmaker makes recurring two-stage decisions through a sequence of discrete-time steps $t \in \{0, 1, \cdots, T\}$. T can be either finite or infinite. Given a partitioned area and time step t, the *first-stage decision* is about whether to hold or match the batched passengers and drivers in each grid; this decision can be made asynchronously in different grids. The *second-stage decision* is to optimize the matching of the batched passengers and drivers in the combined grids where the matchmaker decides to match. The optimization objective of these sequential decisions is to maximize the *total wait reward*[1]. The notation used throughout this work is summarized in Table A.1 in Appendix A.

This sequential decision-making problem can be elaborated as a Markov decision process (MDP). At each time step t, the matchmaker first observes the grid-based state matrix $\boldsymbol{S}(t)$ of the system by gathering information, including currently batched passenger trip details and idle driver locations, as well as predictions about the future supply and demand in each grid. The matchmaker then follows a policy function $\pi(\boldsymbol{A}(t) = a|\boldsymbol{S}(t) = s)$ to make the first-stage decision. All passengers and drivers within the grids with action $\boldsymbol{A}(t) = 0$ ("hold") will be held in the buffer for one more time step. Otherwise, all passengers and drivers within the grids with action $\boldsymbol{A}(t) = 1$ ("match") will be globally matched in the second-stage decision. Assuming that each driver can take at most one passenger request per match (i.e., not considering ride-pooling), we make the second-stage decision by solving a min-cost bipartite matching problem (Williamson and Shmoys, 2011). At the end of these two-stage decisions, unless abandoning the queue, unmatched drivers or passengers will circulate back to the buffer and new arrivals will be admitted into the buffer. Further, the system's state will transition to $\boldsymbol{S}(t+1)$ accordingly.

If we define the wait reward incurred as $R(t) = r(\boldsymbol{S}(t), \boldsymbol{A}(t), \boldsymbol{S}(t+1))$ and a state transition probability $p(\boldsymbol{S}(t+1)|\boldsymbol{S}(t), \boldsymbol{A}(t))$, the

[1] To align with the reinforcement learning terminology, we use the term "wait reward", the negated wait cost, and its derivatives such as reward signal, reward design, and sparse reward issue when dealing with the RL modelling, while in other cases, we use the more intuitive term "wait cost". To this end, maximizing the wait reward and minimizing the wait cost become the equivalent objectives of the optimization.

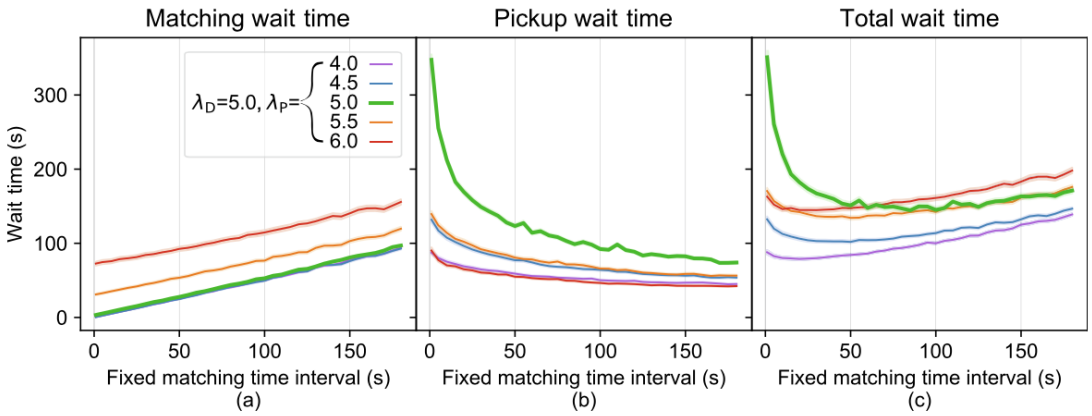

Fig. 2. Change of passenger wait times as the matching time interval is extended in a deque model.

expected total wait reward starting from t can be expressed as $V_\pi(\boldsymbol{S}(t)) = \mathbb{E}_\pi[\sum_{k=t+1}^{\infty}\gamma^{k-t-1}R(k)]$, where γ is a discount factor. Then the matchmaker's objective is to find an optimal policy π^* to maximize the expected total reward $V_\pi(\boldsymbol{S}(0))$,

$$\max_\pi \quad V_\pi(\boldsymbol{S}(0)) = \mathbb{E}_\pi[\sum_{t=1}^{\infty}\gamma^{t-1}R(t)] \tag{1}$$

$$= \sum_a \pi(a|\boldsymbol{S}(0))\sum_{s'}[r(\boldsymbol{S}(0),a,s') + V_\pi(s')]p(s'|\boldsymbol{S}(0),a),$$

where the second summation term can be denoted as an action value function $q_\pi(s,a)$,

$$q_\pi(s,a) := \sum_{s'}[r(s,a,s') + V_\pi(s')]p(s'|s,a). \tag{2}$$

If all the aforementioned functions and their parameters are *known*, π^* can be exactly solved, regardless of the large state space of $\boldsymbol{S}(t)$. However, the formulation above deems intractable in practice. The reasons are twofold:

a. As we have no prior knowledge of either the transition probability $p(s'|s,a)$ or the reward function $r(s,a,s')$, exact solutions are ruled out for this problem.
b. Aggregation of the original state space matrix $\boldsymbol{S}(t)$ to some smaller space produces partially observed state, which may violate the Markov assumption in MDP[2].

These two challenges suggest that finding an optimal on/off decision policy for an online passenger-driver matchmaking system is by no means trivial. Next sections will elaborate on how they are approached.

3.2. Preliminaries on the matching delay trade-offs

This subsection explains how the matching time interval affects the matching wait time and the pickup wait time in a ride-hailing system. We introduce a deque model to approximate the matching process (Xu et al., 2020b). Due to deque's conciseness in theory and descriptiveness in practice, we can derive its performance metrics accordingly.

Without loss of generality, we assume passengers and idle drivers arrive by two independent stationary Poisson processes (arrival rate λ_P and λ_D resp.). Their locations are uniformly distributed within a square area (size of $d \times d$). Note that the choice of demand and supply models does not affect the proposed RL algorithm in the following section. Newly arrived passengers and drivers will balk or renege the queue if the expected wait time exceeds a threshold t_{max}. Upon their arrival, both passengers and drivers will first be batched in a buffer pending matching. Pickup wait times are estimated by the Manhattan distance between passengers and drivers divided by the average pickup speed v. Unmatched passengers or drivers will circulate back to the buffer and be carried over to the next matching time window. A *fixed* matching time interval strategy matches passengers and drivers in a bipartite graph in the second stage at each of a sequence of discrete-time steps $t \in \{\Delta t, 2\Delta t, 3\Delta t, \cdots\}$. This method is suitable if deque arrivals are stationary, in which the delayed matching problem is equivalent to finding an optimal fixed Δt. If the arrivals are non-stationary, optimizing *adaptive* Δt's becomes substantially more challenging.

Fig. 2 illustrates the trade-off curves derived via simulation from the deque model under scenarios with different supply (λ_D) and

[2] The transformation of state space may retain the Markovian properties under conditions, and interested readers are referred to Proposition 3 in Kim and Smith (1995).

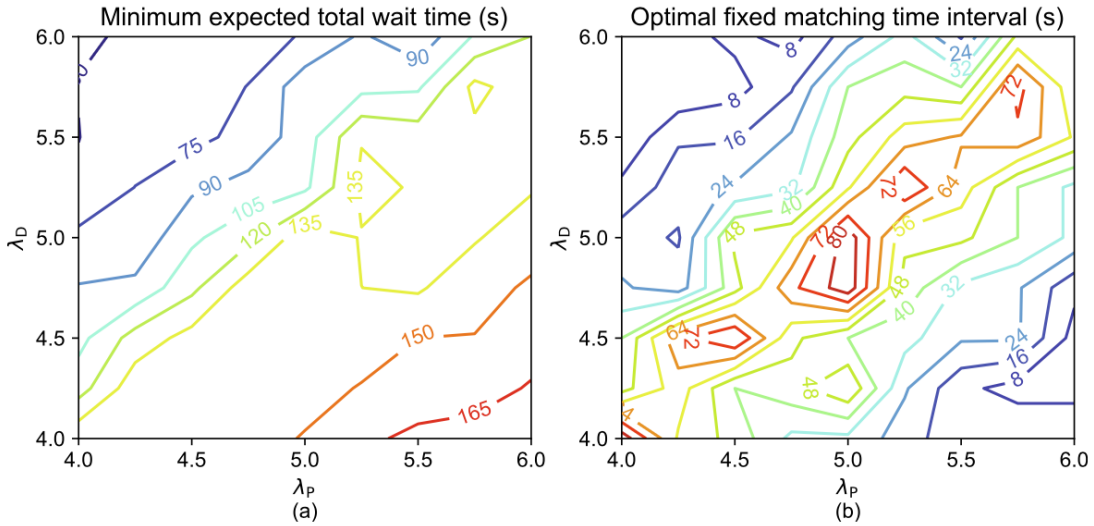

Fig. 3. Minimum expected total wait time and its corresponding optimal fixed matching time interval.

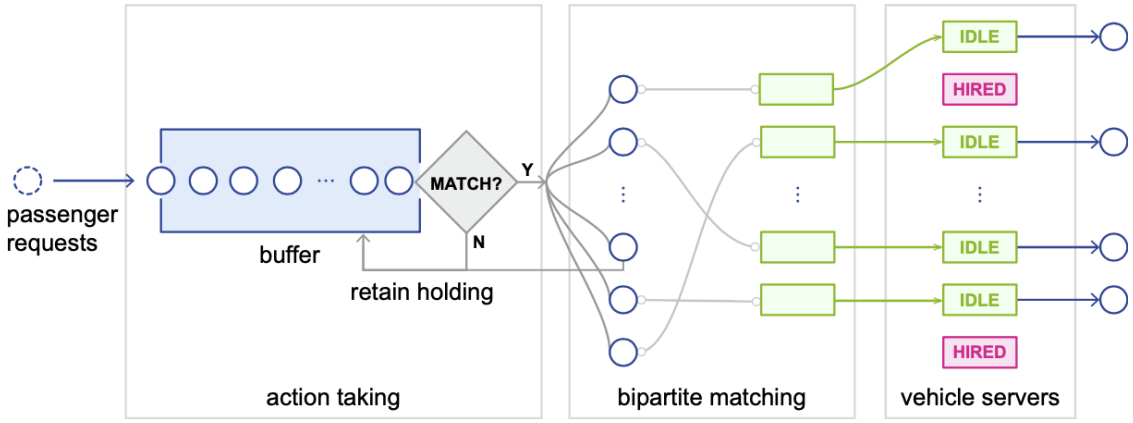

Fig. 4. Workflows of the online ride-hailing simulator.

demand (λ_P). As the matching time interval Δt extends, the matching wait time increases in a approximately linear fashion, as shown in Fig. 2(a), while the pickup wait time curves in Fig. 2(b) descend more steeply. In particular, the scenarios with balanced supply and demand have the steepest U-shape trade-off curves in Fig. 2(c). These trade-off curves empirically manifest the considerable potential of implementing delayed matching in ride-hailing services.

Contour plots in Fig. 3 further depict the minimum expected total wait time and its corresponding optimal Δt, which can be regarded as a value function and a policy function under stationary arrivals. Fig. 3(a) shows that the minimum expected total wait time increases as the gap between λ_P and λ_D broadens, while the minimum wait times are similar if the difference between λ_P and λ_D is held constant. Fig. 3(b) shows that the optimal matching intervals must be substantially prolonged if λ_P and λ_D are balanced, otherwise they should have a shorter delay. The contour patterns clearly corroborate the necessity of delayed matching, especially when $\lambda_P \approx \lambda_D$.

In the next subsection, we will generalize the deque model to a ride-hailing simulator as a dock for accommodating non-stationary and spatially-dependent arrivals. The rest of the current work is underpinned by this simulator.

3.3. A queueing-based ride-hailing simulator

This section adopts a RL approach that approximates the transition dynamics in the MDP to resolve the first challenge in the original formulation. A ride-hailing system simulator enabling the RL approach to sample sequences of states, actions, and matching rewards is needed. To be specific, a ride-hailing simulator provides a controllable interface that connects RL agents to an approximate real-world ride-hailing system. On the real-world side, the simulator is required to simulate trip request collection, delayed matching, and delivery of passengers. On the learning agents' side, the simulator is required to generate observations $\boldsymbol{O}(t)$ that characterize the ride-hailing system's state $\boldsymbol{S}(t)$ and provide informative reward signals. In this regard, we decompose the simulator creation task into two parts: (a) developing an online ride-hailing simulator and (b) designing observations and reward signals that assist in agent learning.

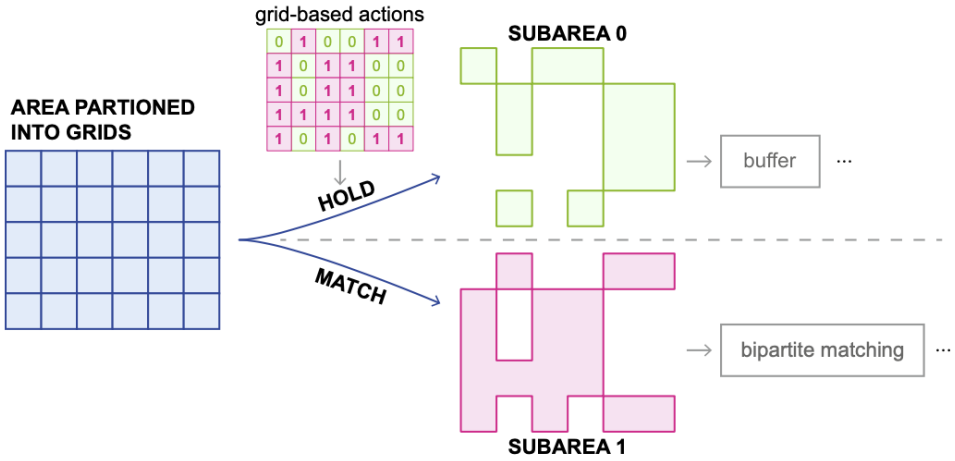

Fig. 5. Grid-based actions and subarea-based matching in the ride-hailing simulator.

3.3.1. Creating an interactive simulator for RL

Our first attempt is to reduce the environment complexity by modeling the arrival of the passenger requests and idle drivers as a multi-server queueing process, as shown in Fig. 4.

Arriving passenger requests are first admitted into a *buffer*. Apropos of action matrix $A(t)$, actions will be taken separately in two subareas (illustrated in Fig. 5), where the subarea containing all the grids whose $A(t) = 0$ is denoted as $\mathbb{S}_0$, and the subarea containing all the grids whose $A(t) = 1$ as $\mathbb{S}_1$. The buffer will retain holding requests in the subarea $\mathbb{S}_0$ while releasing the batched requests in the subarea $\mathbb{S}_1$ to the *bipartite matching* module. Note that partitioning the area and expanding the one-value action to a mixed-action matrix allows the agents to accommodate spatially dependent arrivals. Once the bipartite matching module receives requests, it will pull all idle drivers and execute a global bipartite matching between these two sides in $\mathbb{S}_1$. For simplicity, we estimate the edge cost by the Manhattan distance between passengers and drivers divided by the average pickup speed v and solve for the minimum pickup wait time. Those requests matched are forwarded to the corresponding *vehicle servers* module, while those unmatched requests return to the buffer.

To be specific, the second stage bipartite matching is solved as a linear sum assignment problem (LSAP). LSAP is modeled as

$$\text{Minimize}: \quad \sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}x_{ij} \tag{3}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,2,\cdots,n)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1,2,\cdots,n)$$

$$x_{i,j} \in \{0,1\} \qquad (i,j = 1,2,\cdots,n),$$

where T_{ij} is the pickup time between driver $i \in (1,2,\cdots,N_D)$ and passenger $j \in (1,2,\cdots,N_P)$ estimated by their Manhattan distance divided by the average pickup speed v. In case $N_D \neq N_P$, T_{ij} will be filled into an $n \times n$ matrix with a very large value M, where $n = \max(N_D, N_P)$. Namely, $T_{ij} = M$ for $i \in (N_D + 1, \cdots, n)$ or $j \in (N_P + 1, \cdots, n)$. In the ridepooling setting, the second stage problem is to solve a general assignment problem (Alonso-Mora et al., 2017), and the queueing model needs to be modified to be a matching queue (Cao et al., 2020). Note that these extensions do not affect the general learning framework.

Simulating idle drivers' repositioning decisions over these grids in a multi-server queue is critical for the accuracy of the simulator. This work assumes that drivers stay in the same grid, since each grid's size is relatively large compared to the idle time. In addition, the simulator uses a fixed patience time for passengers throughout this study and assumes that drivers always stay active in the system. These features can be easily extended and embodied in the simulator with new data access. The key motivation for developing such a ride-hailing simulator is to provide an interface for agents to evaluate and improve their policy $\pi(a|s)$, which can been seen as a special case of the model-based RL (Sutton and Barto, 2018).

4. Methodology

This section first describes the design of states and reward signals of the ride-hailing environment, which provide essential information for RL agents to improve their policies. It then introduces a family of state-of-the-art RL algorithms called policy gradient methods. These methods take advantage of the special structure of the delayed matching problem and navigate the sampled trajectories to approximate the value and policy functions. To further improve the stability of the algorithm in a dynamic ride-hailing system, we introduce variants of the Actor-Critic method to solve for the near-optimal delayed matching policy with streaming real-world data.

4.1. Designing observable states and reward signals of the ride-hailing environment

4.1.1. Observable states

A state can be treated as a signal conveying information about the environment at a particular time (Sutton and Barto, 2018). In a ride-hailing environment, a complete sense of "how the environment is" should include full information consisting of, but not limited to, timestamp and origin-destination (OD) of all the batched requests, current locations of drivers, and predictions about the future supply and demand. However, the complete state space of the original ride-hailing system is enormous and sampling from it efficiently becomes impossible. Inspired by the analytical models built in the previous literature (Yang et al., 2020), we abstract aggregated variables from the ride-hailing simulator as partial observations of the environment. The observations are concatenated into a tuple as

$$\boldsymbol{O}(t) = \{(N_{\mathrm{P}}(i,t), N_{\mathrm{D}}(i,t), \lambda_{\mathrm{P}}(i,t), \lambda_{\mathrm{D}}(i,t)) | i \in \mathbb{S}\}, \tag{4}$$

where $N_{\mathrm{P}}(i,t)$ (resp., $N_{\mathrm{D}}(i,t)$) is the number of batched passenger requests (resp., idle drivers) in grid i at time step t, and $\lambda_{\mathrm{P}}(i,t)$ (resp., $\lambda_{\mathrm{D}}(i,t)$) is the estimated arrival rate of passenger requests (resp., idle drivers) in grid i from t onward.

Aggregating the complete state space $\boldsymbol{S}(t)$ into the observation $\boldsymbol{O}(t)$ will generalize the previously formulated MDP to a partially observable Markov decision process (POMDP), which echos the second challenge aforementioned. We can assume an unknown distribution $p(\boldsymbol{O}(t)|\boldsymbol{S}(t))$ linking the complete state $\boldsymbol{S}(t)$ and a partial state $\boldsymbol{O}(t)$ observed from the deque model. The matchmaker can re-establish the value function and optimization objective in this setting by replacing $\boldsymbol{S}(t)$ in the MDP setting with $\boldsymbol{O}(t)$. In addition, stochastic policies are enforced to obtain a higher average reward than a deterministic policy (Jaakkola et al., 1995).

Of all the variables in the observation, $\lambda_{\mathrm{P}}(i,t)$ is the only variable that is not directly collectable from the simulator. The learner uses either the average recent arrival rates as a proxy or a more sophisticated predictor to estimate future demand. In contrast, $\lambda_{\mathrm{D}}(i,t)$ is easy to obtain as the simulator continues to track vehicle locations in real time. The lost information of exact locations of vehicles and passengers only affects the first-stage decision $\boldsymbol{A}(t)$. At the second stage, this information has been considered in the edge cost in the bipartite matching and included in the reward function. In summary, we reduce the state space of $\boldsymbol{S}(t)$ significantly by transferring partial information noise in $\boldsymbol{O}(t)$ of the rewarding process.

With observable queueing-based states, the state transition dynamics, i.e., how $\boldsymbol{O}(t)$ should be updated after taking an action, is straightforward:

$$\begin{cases} N_{\mathrm{P}}(i,t) = N_{\mathrm{P}}(i,t-1) + n_{\mathrm{P}}(i,t-1:t) \\ N_{\mathrm{D}}(i,t) = N_{\mathrm{D}}(i,t-1) + n_{\mathrm{D}}(i,t-1:t) \end{cases}, i \in \mathbb{S}_0 \quad \text{(hold)} \tag{5}$$

$$\begin{cases} N_{\mathrm{P}}(i,t^-) = N_{\mathrm{P}}(i,t-1) + n_{\mathrm{P}}(i,t-1:t) \\ N_{\mathrm{D}}(i,t^-) = N_{\mathrm{D}}(i,t-1) + n_{\mathrm{D}}(i,t-1:t) \\ N_{\mathrm{M}}(i,t) = \min(N_{\mathrm{P}}(i,t^-), N_{\mathrm{D}}(i,t^-)) \\ N_{\mathrm{P}}(i,t) = N_{\mathrm{P}}(i,t^-) - N_{\mathrm{M}}(i,t) \\ N_{\mathrm{D}}(i,t) = N_{\mathrm{D}}(i,t^-) - N_{\mathrm{M}}(i,t) \end{cases}, i \in \mathbb{S}_1 \quad \text{(match)} \tag{6}$$

where $n_{\mathrm{P}}(i,t-1:t)$ and $n_{\mathrm{D}}(i,t-1:t)$ are the number of trip requests and idle drivers arriving in grid i between $t-1$ and t, respectively. $N_{\mathrm{P}}(i,t^-)$ and $N_{\mathrm{D}}(i,t^-)$ are the number of requests and drivers updated before being matched at t, as they may change during the time step. Meanwhile, $\lambda_{\mathrm{P}}(i,t)$ and $\lambda_{\mathrm{D}}(i,t)$ are continuously re-estimated from historical passenger arrivals and real-time vehicle locations accordingly.

4.1.2. Reward signals

A reward signal is the environment's valuation of an implemented action $\boldsymbol{A}(t)$. In the context of supervised learning, this reward signal is required to instruct what action the agents should take; however it is not required in RL. To this end, a major advantage of RL distinguishing itself from its counterpart is that the instruction for optimal actions is not mandated.

Nevertheless, the reward signals still must be carefully designed, especially when the environment faces sparse rewards in this setting (i.e., the reward is 0 most of the time in delayed matching). However, a *naïve* reward signal will remain unknown until a match is executed and the total wait cost for all served passengers is determined. In this regard, the buffer that continues to hold upcoming requests receives zero rewards. This sparse reward issue makes it extremely difficult for RL to relate a long sequence of actions to a distant future reward (Hare, 2019). In this ride-hailing environment, the agents fed with this naïve reward are stuck taking $a = 0$, since the zero reward of taking $a = 0$ is greater than any negative rewards incurred by taking $a = 1$.

To tackle this challenge, we try to decompose this one-shot reward happening at a certain time into each time step within the passengers' queueing lifetime. Since the total matching wait time is in itself a summation of the wait time at each time step, it is straightforward to delineate its increment as a matching wait reward:

$$R_{\mathrm{m}}(t) = -\sum_{i \in \mathbb{S}} [N_{\mathrm{P}}(i,t-1) + \tau_{\mathrm{m}}(n_{\mathrm{P}}(i,t-1:t))], \tag{7}$$

where $\tau_{\mathrm{m}}(\cdot)$ computes the matching wait time of the newly-arrived requests in grid i from $t-1$ to t. In particular, if the arrival of passengers is assumed to be a Poisson process, $\tau_{\mathrm{m}}(n_{\mathrm{P}}(i,t-1:t)) \approx n_{\mathrm{P}}(i,t-1:t)/2$.

However, the pickup wait time is not a summation per se; its decomposition is less straightforward. To resolve this issue, we devise an artifice so that these increments add up to the final pickup wait time in the matching process. As shown in Fig. 6, the intuition is to decompose the one-shot pickup wait time $\tau_{\mathrm{p}}(t)$ to a summation of step-wise incremental pickup wait times, i.e., $\tau_{\mathrm{p}}(1)$, ⋯,

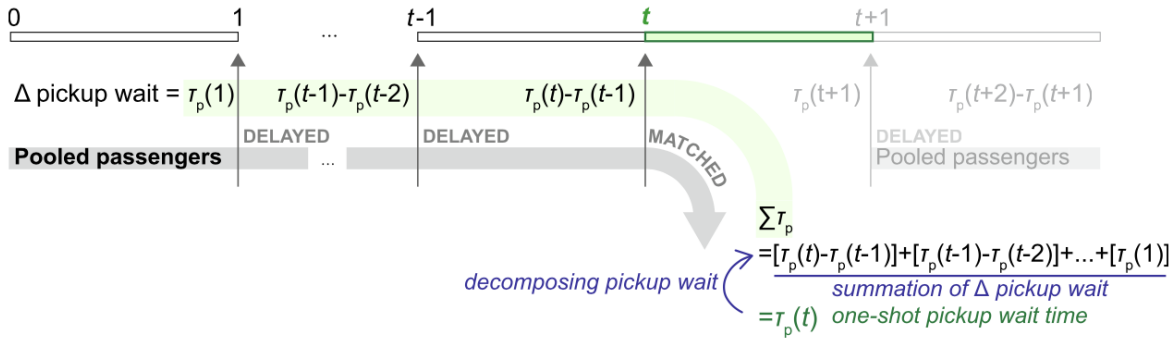

Fig. 6. Decomposition of the one-shot pickup wait time $\tau_p(t)$ to a summation of step-wise incremental pickup wait times.

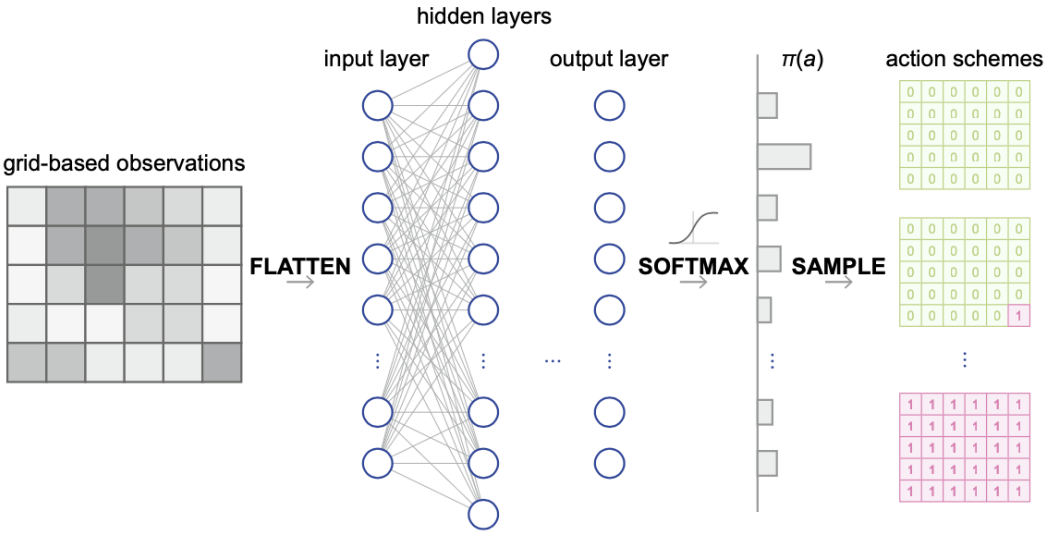

Fig. 7. Structure of the policy neural network.

$\tau_p(t-1)-\tau_p(t-2), \tau_p(t)-\tau_p(t-1)$.

In light of this decomposition, we define the step reward mathematically as:

$$R_p(\mathbb{S},t)=\begin{cases}-\left[\tau_p(N_P(\mathbb{S}_0,t),N_D(\mathbb{S}_0,t))-\tau_p(N_P(\mathbb{S}_0,t-1),N_D(\mathbb{S}_0,t-1))\right], & \mathbb{S}=\mathbb{S}_0\\ -\tau_p(N_P(\mathbb{S}_1,t),N_D(\mathbb{S}_1,t)), & \mathbb{S}=\mathbb{S}_1\end{cases} \tag{8}$$

where $\tau_p(N_P(\mathbb{S}_0,0),N_D(\mathbb{S}_0,0)):=0$, and

$$N_P(\mathbb{S}_0,t)=\sum_{i\in\mathbb{S}_0}N_P(i,t),\quad N_P(\mathbb{S}_1,t)=\sum_{i\in\mathbb{S}_1}N_P(i,t),$$

$$N_D(\mathbb{S}_0,t)=\sum_{i\in\mathbb{S}_0}N_D(i,t),\quad N_D(\mathbb{S}_1,t)=\sum_{i\in\mathbb{S}_1}N_D(i,t).$$

Note that the difference in Eq. 8 is the net increase of the pickup wait reward if holding ($A(\mathbb{S}_0,t)=0$) requests for one more time step. Comparing this net increase with the net loss of the matching wait reward within the same time step can indicate whether holding the passenger requests is a better option. Thus, this decomposition not only addresses the sparse reward issue but also introduces a short-term signal to facilitate the learning process.

Therefore, the total pickup wait reward at t is

$$R_p(t)=R_p(\mathbb{S}_0,t)+R_p(\mathbb{S}_1,t). \tag{9}$$

Summing up Eqs. 7 and 9, we obtain the total wait reward at time t:

$$R_w(t)=c_mR_m(t)+c_pR_p(t), \tag{10}$$

where c_m and c_p are the perceptual cost per unit wait time. These weights can be set to accommodate the asymmetrical perception of time values in different stages of the waiting process (Guo et al., 2017). In general, $c_m \gg c_p$ as passengers are less patient in the face of uncertain matching outcome (Xu et al., 2020a).

4.2. Learning optimal delayed matching policy through Actor-Critic methods

The intricacy of applying RL methods to a realistic ride-hailing environment provokes the need to appropriately adapt the general-purpose RL methods for improved robustness. On account of this, we present two policy gradient methods along with three aspects of meticulous adaptation.

First, outputting multi-binary actions. The actions taken in the conventional RL setting are mostly discrete or continuous scalars, occasionally being continuous vectors, while the action space in the ride-hailing environment, which is defined over a spatial network, is a multi-binary matrix. To resolve this difficulty of representation, we propose a set of *action schemes* $\mathbb{A}$, as presented in Fig. 7, to map the one-dimensional action scheme ID (ranging from 1 to $2^{|\mathbb{S}|}$) chosen by the agent to the action matrix (ranging from a matrix of zeros to ones) for feeding the environment. Similarly, we obtain the probability of choosing to match in each grid by summing the learned policy probability $\pi(\boldsymbol{A}|s)$ of choosing each action scheme.

$$p(i) = \sum_{\boldsymbol{A}\in\mathbb{A}} \pi(\boldsymbol{A}|s) \cdot 1_{\boldsymbol{A}(i)=1}, \tag{11}$$

where $1_{A(i)=1}$ is an indicator function that equals 1 if the action in grid i, i.e., $\boldsymbol{A}(i)$, is 1 (match), and equals 0 otherwise.

Second, increasing sample efficiency and reducing sample correlation. In the standard RL setting, the agent's experiences (observations, actions, and rewards) are usually utilized once in value and policy updates. In the light of the large dimensionality of observation $\boldsymbol{O}(t)$, this one-time utilization can amount to a substantial waste of rare experiences and hinders the rate of learning. We therefore adopt a technique called *experience replay* (Foerster et al., 2017) to enable reuse of sampled experiences. Additional usefulness of experience replay is the reduction of sample correlation. More specifically, the sampled observations are step-wise correlated because each observation is updated upon the previous step's carryover of supply and demand. Experience replay blends the action distribution over the agent's previous observations, which can mitigate the divergence in parameters (Mnih et al., 2013).

Third, training agents with specific time limits. The interaction between an agent and the ride-hailing environment has no terminal condition in standard RL literature. However, the ride-hailing system's matching policy requires setting time limits at the end of each sampling period to smooth out the training processes. This coincides with the notion of episodes in the experience sampling. It is worth noticing that, if the time limits are regarded as terminal conditions, the RL algorithm will render an incorrect value function update. To distinguish time limits from terminal conditions, we must perform a *partial episode bootstrapping* to continue bootstrapping at the time limit. With this treatment, agents can learn the delayed matching policy beyond the time limit (Pardo et al., 2018).

In the next subsections, we start by briefly introducing the general concept of policy-gradient methods. Then, we describe two policy gradient-based methods, Actor-Critic and ACER, in which ACER is our solution to addressing those issues mentioned above.

4.2.1. Overview of policy gradient methods

Policy gradient methods are a family of RL methods that skip consulting the action values and learn a parameterized policy function $\pi_{\boldsymbol{\theta}}(a|s)$ directly. In general, they have many advantages over value-approximation methods, including stronger convergence guarantees and the capacity of learning a stochastic policy (Sutton and Barto, 2018). As discussed in the section of designing observable states, using a stochastic policy can also obtain higher average rewards than deterministic ones in POMDP.

Policy gradient methods define a performance measure $J(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ denotes the parameterization of the policy in terms of the observable state, and improves it along the direction of gradient ascent. The performance measure of a given policy is formally defined as

$$J(\boldsymbol{\theta}) = V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{O}(0)),$$

where the right-hand side value function denotes the expected return starting from state $\boldsymbol{O}(0) = s$ by following sequences of actions taken from the policy $\pi_{\boldsymbol{\theta}} := \pi_{\boldsymbol{\theta}}(a|s)$.

The state value function $V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{O}(0))$ depends on both the policy and its changes on the ride-hailing environment's state distributions. However, the latter effect is unknown. As a result, it is challenging to estimate $\nabla J(\boldsymbol{\theta})$. Fortunately, a solution in the form of the policy gradient theorem has circumvented the derivative of the state distributions and provided an analytic expression for $\nabla J(\boldsymbol{\theta})$,

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \eta(s) \sum_a q_\pi(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s) = \mathbb{E}_\pi \left[\sum_a q_\pi(\boldsymbol{O}(t),a) \nabla \pi_{\boldsymbol{\theta}}(a|\boldsymbol{O}(t)) \right]. \tag{12}$$

where q_π is the true action-value function for policy π, and $\eta(s)$ is the on-policy distribution of s under policy π.

Following the gradient ascent method, the policy parameters are updated by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \sum_a \widehat{q}(\boldsymbol{O}(t),a) \nabla \pi_{\boldsymbol{\theta}}(a|\boldsymbol{O}(t)),$$

where $\widehat{q}$ is the learned approximation to q_π, and α is the learning rate.

Replacing a in Eq. (12) with $\boldsymbol{A}(t)$, we derive a vanilla policy gradient algorithm called REINFORCE whose $\nabla J(\boldsymbol{\theta})$ can be computed by

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi [G(t) \nabla \ln \pi_{\boldsymbol{\theta}}(\boldsymbol{A}(t)|\boldsymbol{O}(t))],$$

where $G(t)$ is the total matching wait reward from t onward.

4.2.2. Actor-Critic and ACER

Unlike REINFORCE and its variant methods, Actor-Critic estimates the total return $G(t)$ based on a bootstrapping, i.e., updating the value estimation of the current state from the estimated value of its subsequent state,

$$G(t) := V_{\mathbf{w}}(\boldsymbol{O}(t)) \leftarrow R(t) + \gamma V_{\mathbf{w}}(\boldsymbol{O}(t+1)). \tag{13}$$

Bootstrapping the state value function $V_{\mathbf{w}}(\cdot)$ (termed "Critic") introduces bias and an asymptotic dependence on the performance of the policy function (termed "Actor") estimation. This is inversely beneficial for reducing the variance of estimated value function and accelerates learning. To meet the needs of training in the time-limit ride-hailing environment, we adapt the Actor-Critic method, as summarized in Algorithm 1, to bootstrap at the last observation in each episode by replacing the update on Line 6 with $R + \gamma V_{\mathbf{w}}(O^{'})$.

Algorithm 1. The Actor-Critic method

Input: $\pi_\theta(a|s)$, $V_{\mathbf{w}}(s)$, step size $\alpha^\theta > 0$, $\alpha^{\mathbf{w}} > 0$,discount factor $\gamma \in [0,1]$
1 Initialize policy parameter $\theta \in \mathbb{R}^d$, and state-value weights $\mathbf{w} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$);
2 **for each episode do**
3 Initialize O (initial state of the episode), $I \leftarrow 1$
4 **while** O **is not terminal do**
5 $A \sim \pi_\theta(\cdot|O)$, take action A, observe O', R
6 $\delta \leftarrow [R + \gamma V_{\mathbf{w}}(O')] - V_{\mathbf{w}}(O)$
7 $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla V_{\mathbf{w}}(O)$
8 $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi_\theta(A|O)$
9 $I \leftarrow \gamma I, O \leftarrow O'$
10 **end**
11 **end**

ACER, standing for Actor-Critic with experience replay, is an upgrade to Actor-Critic by enabling reuse of sampled experience (Wang et al., 2016). It integrates several recent advances in RL to help stabilize the training process in the intricate ride-hailing environment. More specifically, ACER is a modification of Actor-Critic by adding a buffer to store past experiences step by step for later sampling to batch update the value and policy neural networks. Some other techniques, such as truncated importance sampling with bias correction, stochastic dueling networks and an efficient trust region policy optimization method, are also added to enhance the learning performance. The proposed ACER method is also adapted to the time-limit ride-hailing simulator by enabling bootstrapping at the last observations. Since the entire algorithm is rather intricate and out of the scope of this study, we just present a general framework of ACER in Algorithm 2 to give a broad picture of how it works.

Algorithm 2. Actor-Critic with experience replay (ACER)

Input: Dense NN Actor$(a|s)$, Critic(s), and TargetCritic(s), discount factor $\gamma \in [0,1]$, experience replay batch size N, delayed update rate τ
for each episode do
  Initialize O (initial state of the episode), buffer
  while O **is not terminal do**
  $A \sim$ Actor$(\cdot|O)$, take action A, observe O', R from the ride-hailing simulator
  buffer.push(O, A, R, O') `// Collect experience`
  `// On-policy update`
  Follow Actor-Critic update
  $O \leftarrow O'$
  end
  `// Off-policy update`
  if buffer.size $> N$ **then**
  $O, A, R, O' =$ buffer.sample(N) `// Sample experience to replay`
  $\delta \leftarrow [R + \gamma$TargetCritic$(O')] -$ Critic(O)
  CriticLoss $\leftarrow \delta^2$, ActorLoss $\leftarrow \delta \nabla \ln$ Actor$(A|O)$
  CriticLoss.minimize(), ActorLoss.minimize()
  TargetCritic.$\mathbf{w}' = (1-\tau)$TargetCritic.$\mathbf{w}' + \tau$Critic.$\mathbf{w}$ `// Delayed update`
  end
end

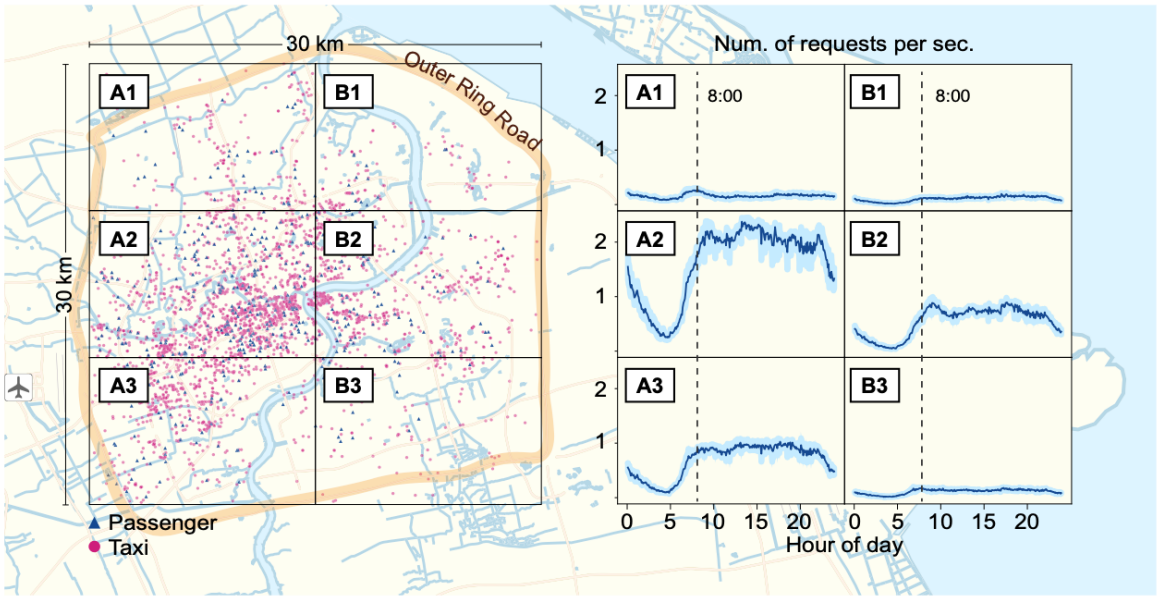

Fig. 8. Spatial and temporal distributions of taxi supply and demand in downtown Shanghai, China.

4.2.3. *Baseline: fixed matching time interval strategy*

Industries currently implement a fixed matching time interval throughout an extended period. More specifically, the matchmaker processes batched passenger requests every n time steps. A batch-learning version for the fixed strategy is to run the experiments repeatedly to generate sufficient samples for each matching time interval candidate and choose the one that yields the minimum mean wait costs. The chosen fixed matching time interval strategy is a useful baseline, especially when the arrival of passenger requests is spatiotemporally homogeneous.

It is worth mentioning that this comparison is unfair for the RL-policy as we assume no prior knowledge is available at the beginning of the learning process. On the contrary, the baseline has found the optimal fixed matching time interval using the historical data. A fixed matching time interval can offer a reasonable upper bound wait cost for reference in a real-world scenario with non-homogeneous arrivals and complicated interdependence.

5. Result

In this section, we first delineate the experimental setup for testing the above-mentioned policy gradient methods, then we present and compare the algorithmic performances of these methods and the baseline strategy. Lastly, we analyze the learned policy and interpret how it outperforms others.

5.1. *Data description and experimental setup*

We conduct a numerical experiment using a large-scale dataset from Shanghai Qiangsheng Taxi Co. The one-week taxi dataset was collected in March 2011. It contains trip data, sequential taxi trajectories, and operational status for nearly 9,000 taxis, with fields including taxi ID, date, time, longitude, latitude, speed, bearing, and status identifier (1 for occupied/0 for idle).

Empirical evidence has shown that over 80% of taxi trips are concentrated in downtown Shanghai outlined by the Outer Ring Road (Qin et al., 2017), which is selected as the area of interest for our experiments. To balance the computational overhead incurred by the dimension of actions, we partition the experiment area into six grids (3×2) as shown in Fig. 8. Experiments with finer grids are observed to increase the training time significantly with marginal improvement on the overall performance. Three different partitioning granularities of action are hereafter implemented and compared (i.e., 1×1,1×2, and 3×2). In such a partitioning, a desirable spatial variation of arrival patterns is well captured, with grid A2 producing high demand, grids A1, B1, and B3 producing low demand, and the remaining grids producing medium demand. The simulation initiates at 8:00 AM, which is the start of the morning peak hours.

The ride-hailing environment is initialized with trip requests and taxi supply sampled from the historical dataset as shown in Fig. 9. Then, the environment warms up for the first 10 min, during which time all passengers and drivers are matched instantaneously. Once the warm-up is complete, the learning agents begin interacting with the environment through the simulator. The agents must match requests coming within the next 10 min, which is considered an *episode*. After every episode, the environment is reset and re-warmed up before the agents resume the learning process.

In terms of the algorithm implementation, we use a library called Stable Baselines (Hill et al., 2018) to train the models.

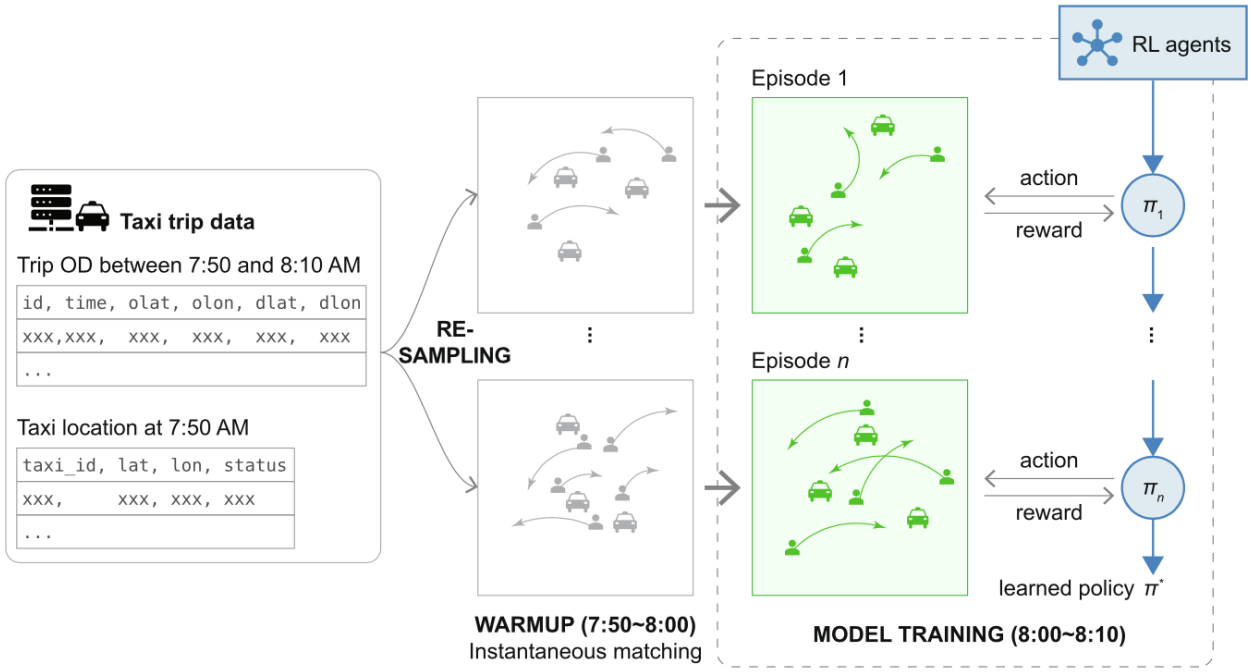

Fig. 9. Experimental setup of the ride-hailing environment and model training.

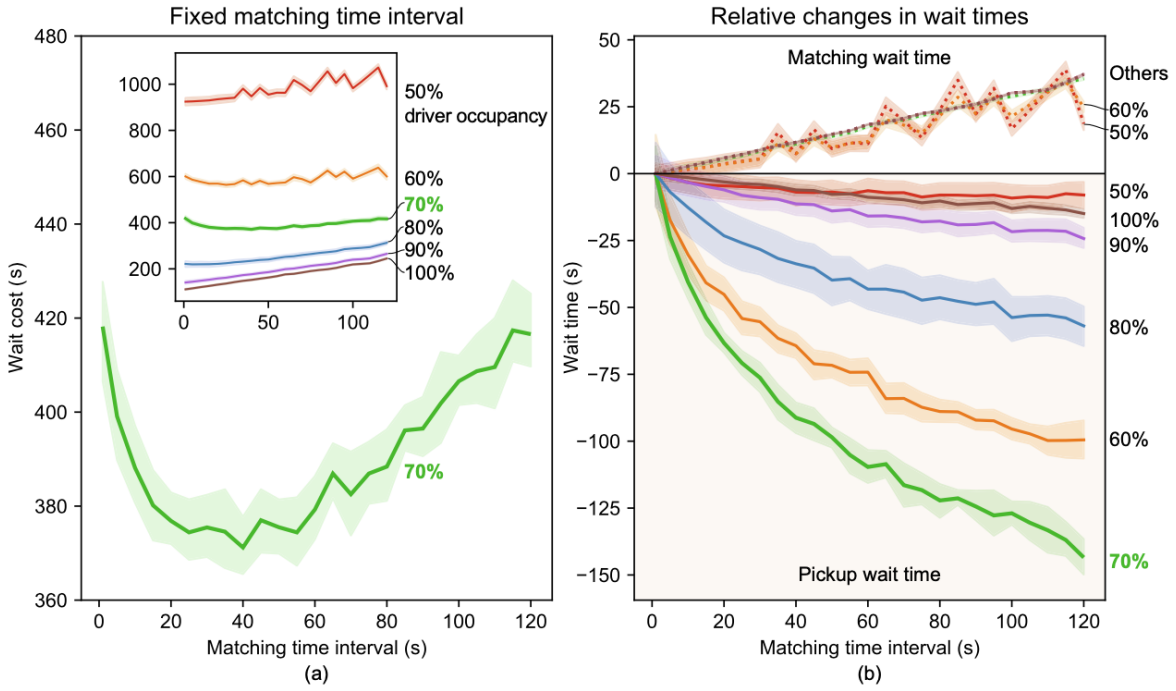

Fig. 10. The relationship between the total wait costs and the matching time interval based on the fixed matching time interval strategy under different driver occupancies: (a) total wait costs and (b) relative changes to the matching and pickup wait times under one-second matching time interval. The lines stand for the mean wait times from experiments and the shaded regions represent 95% confidence interval.

Additionally, we set unit wait cost $c_m = 4$ and $c_p = 1$ to reflect a reasonably higher perceived value of the matching wait time over the pickup wait time, and set average pickup speed $v = 20$ km/h.

5.2. RL algorithm performance

5.2.1. Revisiting trade-offs in delayed matching

As a complement to the previous explanation of matching delay trade-off under stationary arrivals, we now justify how this trade-

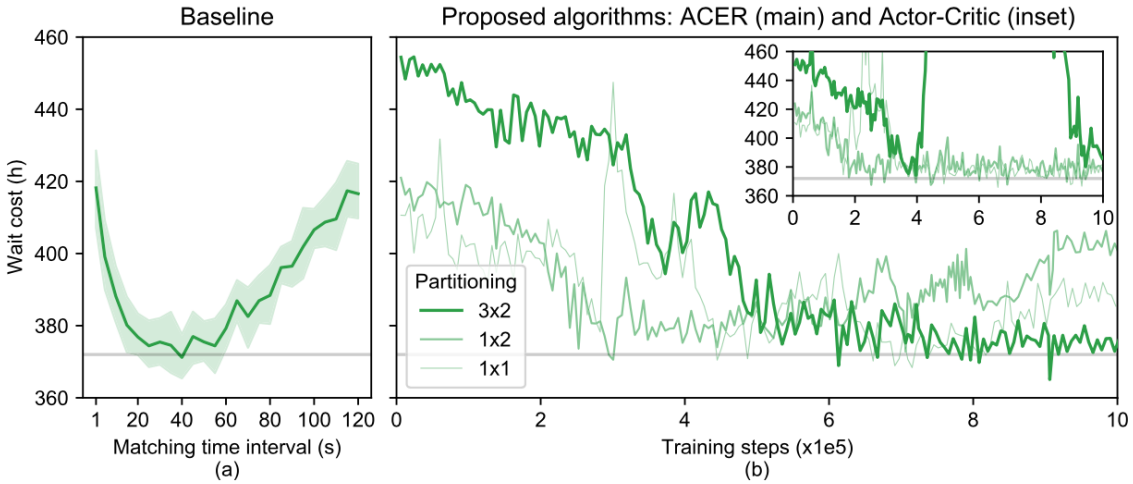

Fig. 11. Performance comparison between (a) the fixed matching time interval strategy and (b) policy gradient algorithms: Actor-Critic with experience replay (ACER, main) and Actor-Critic (inset).

off is established with real-world data. The wait time curves under different driver occupancies are illustrated in Fig. 10(a), where the driver occupancy is a parameter set to mimic a particular supply-demand ratio (100% corresponds to the driver supply observed in the historical data). The result shows that the total wait costs are constantly increasing under either a high ($\geq$ 80%) or low driver occupancy[3] ($\leqslant$50%) (an instantaneous matching is preferred for this case). Under a moderate driver occupancy (70%), the total wait cost indicates a U-shape curve.

Since the vehicle trajectory data is from a traditional street-hailing taxi market with significant empty car mileage, feeding 100% of driver supply into the simulator corresponds to a high-supply scenario. As a result, the search frictions are lower, and the required number of drivers in the context of ride-hailing services should also be smaller. In light of this, the finding here is consistent with the previous results under stationary arrivals – it is necessary and beneficial to perform the delayed matching in the balanced supply-demand scenario.

This trade-off is more comprehensible after further separating the total wait costs into the pickup and matching wait times, as seen in Fig. 10(b). Under different driver occupancies, the relative increases of matching wait times (dotted curves) are almost at a fixed rate, while in contrast, the corresponding decreases of the pickup wait times (solid curves) vary drastically. Such a distinction suggests that the slope of the pickup wait times hugs the U-shape of the matching delay curve. More specifically, this is because the pickup wait times decrease much faster than the fixed increase rate of the matching wait times.

A key implication is that the desirable U-shape of matching delay trade-off merely appears under a specific range of supply-demand ratios. The U-shape trade-off curve's existence condition, from a theoretical perspective, is worth further study; for example, future work could model when an agent should apply instantaneous matching and when the agent should otherwise follow the learned delayed matching policy.

5.2.2. *Algorithmic performance*

Following the discussion above, we find that sampling 70% of the existing taxi drivers is a desirable value to prompt the matching delay trade-off. Using this driver-passenger ratio as the instance for each RL algorithm, we repeat the model training for 1 million steps. The training curves and their comparison with that of the fixed matching time interval strategy are presented in Fig. 11.

In the baseline model, matching every 40 s is found to be optimal, and the served passengers' total wait cost is around 371.20 hours, as seen in Fig. 11(a). Note that the trade-off curve is by no means smooth with real-world data, while the change of wait costs is subject to perturbation of matching time intervals. Learning a near-optimal adaptive delayed matching policy is quite challenging to this end. The proposed algorithm ACER under partitioning 3×2 reports the final converged wait cost, which is on a par with the baseline, while ACERs under partitioning 1×2 and 1×1 fail to converge (Fig. 11(b)). These results suggest that partitioning the area assists the agents in learning better policies. In comparison, the original Actor-Critic algorithm only converges under the partitioning 1×2 to the wait cost around 380 h, however its cost variance is greater than the result of the ACER algorithm.

Table 1 presents the wait time measures of the baseline strategies and the proposed ACER algorithm (AC algorithm is excluded due to its mediocre performance). Compared with the instantaneous matching strategy, the measures show that the learned ACER policy can substantially reduce the mean pickup wait time. This policy can reduce total wait time by 20.41%, which is on a par with previous results in Ke et al. (2020) (292.60 $\rightarrow$ 221.88 : 24.17% reduced). It is worth mentioning that such a comparison with prior work's RL

[3] Under a low driver occupancy ($\leqslant$50%), some passengers experience a long wait beyond their tolerance and choose to renege the queue. Reneging destabilizes the matching wait time in two opposite aspects: first, those reneged passengers have experienced a longer matching wait than otherwise they had been matched in time; second, the matching of the remaining passengers is expedited to some extent. Since the extent of these impacts varies with matching time intervals, both the wait cost (Fig. 10) and the matching wait time (Fig. 10(b)) turn out to be zigzag.

Table 1
Comparison of model performances.

Algorithm	Mean matching wait time (s)	Mean pickup wait time (s)	Mean total wait time (s)
Instant.	3.01	675.71	678.72
Fixed int.	21.20	525.69	546.89
ACER (3 × 2)	26.95	513.22	540.17

Note: "Instant." stands for instantaneous matching, "Fixed int." for fixed matching time interval matching, and ACER for Actor-Critic with delayed update and experience replay. Wait time unit: s/trip request.

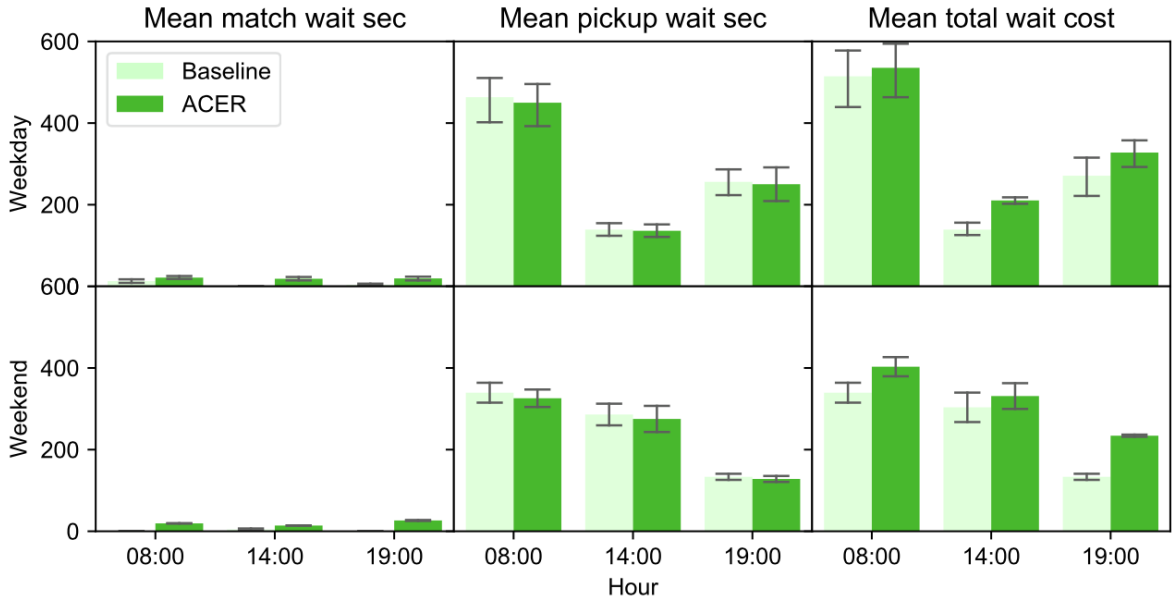

Fig. 12. Testing the learned policy. Baseline: fixed matching time interval strategy. Bar height indicates mean value, error bar 95% confidence interval.

algorithms is unjust, as the environment settings and data sources are different. However, it is plausible to state that, as compared with the multi-agent model in Ke et al. (2020), our single-agent model has a significantly smaller state space, but achieves a similar performance.

In addition, we test the performance of the learned policy for other hours and days in Shanghai. We show that the learned policy is robust within and across days. Specifically, we evaluate the performance of the ACER policy, which is learned with real-world data at 08:00 on a single weekday, in other weekday or weekend hours. The performance is compared with the baseline strategy using fixed matching time interval. As Fig. 12 shows, total wait costs yielded by the ACER policy are close to the baseline's at all weekday 08:00 and weekend 14:00. This indicates that the learned policy generalizes well within and across days of those periods. In the remaining periods, the ACER policy does not generalize well. However, this can be reconciled by taking instantaneous matching because these periods have sufficient supply and hence the instantaneous matching is more desirable. This case can be seen from the near-zero matching wait time in the baseline. This can be an extension of our study to introduce a meta-model that switches the platform's matching policy between instantaneous and delayed matching, mainly affected by the driver occupancy.

To summarize, the proposed two-stage ACER model has demonstrated its capacity to reduce passenger wait costs and improve the quality of ride-hailing services.

5.3. Delayed matching policy interpretation

As the policy neural network is established on a high-dimensional state space, it is impossible to completely visualize the structure of the optimal policy in a low-dimensional space. Instead, we try to interpret the learned policy by analyzing how the matching process unfolds under the learned delayed matching policy. In other words, we replay episodic samples by following the learned policy so as to reveal the decision-making step by step.

To be specific, we will track the matching probability in each grid along the episode to reveal how the learned policy is responding to a certain environment state. Fig. 13 shows the grid-based matching probabilities and their corresponding environment state transitions. It can be seen that the learned delayed matching policy appears to have different time-varying matching probabilities (green curves) in different grids.

As for the three grids on the left (A1, A2 and A3), their matching probability curves look alike on most parts and can be summarized in the following three stages: (a) initial stage (0s to 150s): the system match espassengers and drivers every 2s to 5s on average, since

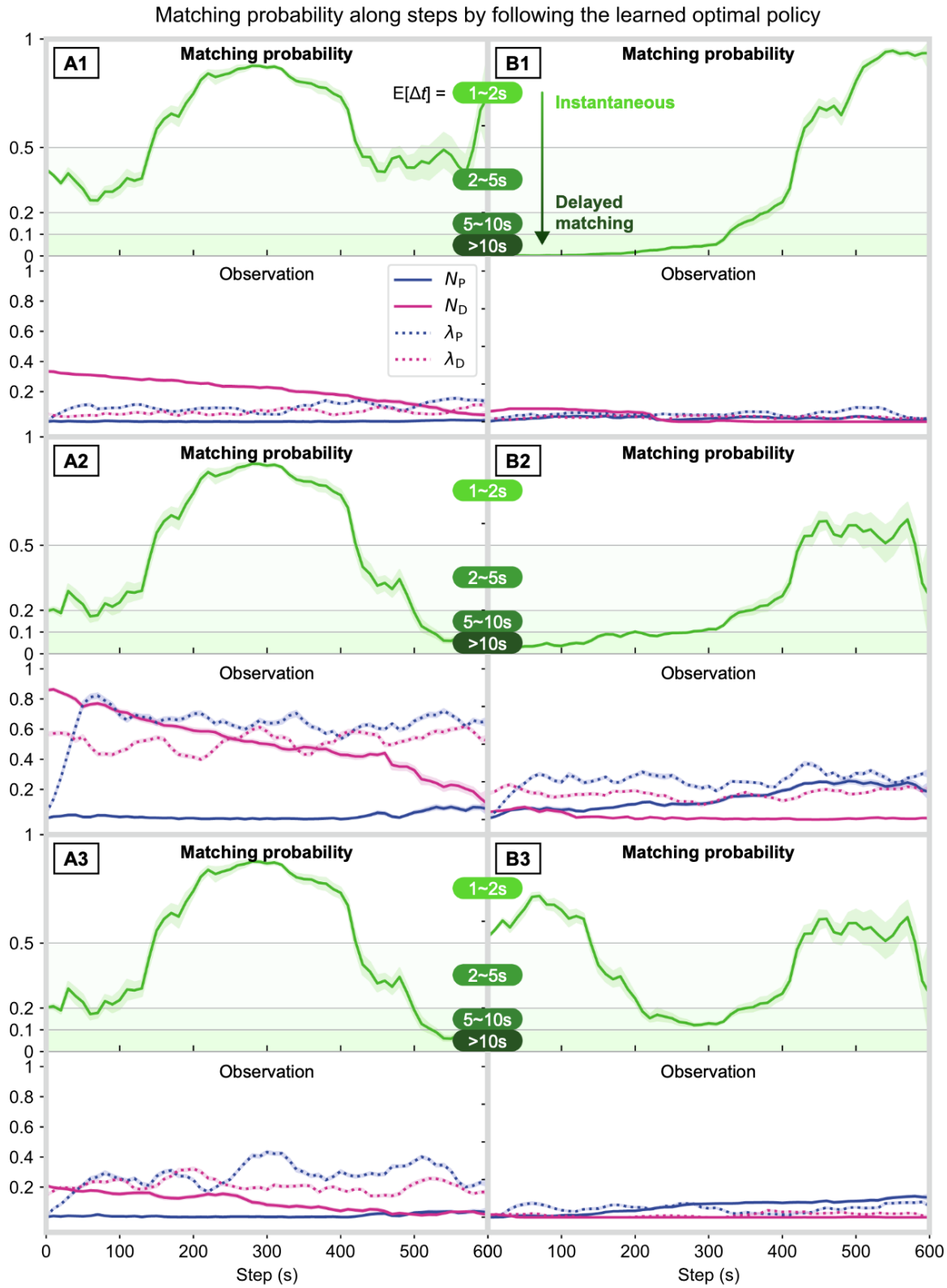

Fig. 13. Episodic replay of a matching process under the learned policy for each grid.

passengers in this period arrive with a shortly increasing estimated rate λ_P (blue dotted curves), a mild delay accumulating the arriving passengers is expected to shorten the pick up distance; (b) stable stage (150s to 450s): the system shortens the matching time interval and make an instantaneous matching every 1s to 2s. Since arrival rates λ_P and λ_D at this stage are stabilized, while the number of available drivers N_D (magenta curves) is remarkably greater than the number of waiting passengers N_P (blue curves), the delayed matching becomes unnecessary and an instantaneous matching is called for; and (c) final stage (450s to 600s): A1 first transitions to 2s to 5s and then returns to 1s to 2s, since its $N_D \gg N_P$ considering the lower level of both λ_P and λ_D, an instantaneous matching is still desirable. By contrast, both A2 and A3 extend their matching time interval beyond 10s, since their $N_D \approx N_P$ considering the higher level of both λ_P and λ_D, a delayed matching is expected under such a condition.

As for the three grids on the right (B1, B2 and B3), their matching probability curves can be summarized in two stages: (a) initial stage (0s to 300s): both B1 and B2 begin matching with a matching time interval beyond 10s, intending to accumulate available drivers to shorten the pickup distance, while B3 makes an instantaneous matching due to very few passengers and driver expected to arrive into this grid; and (b) final stage (300s to 600s): all of the grids choose to make a matching every 1s to 2s, because $N_P \gg N_D$ considering the low level of λ_P and λ_D at this stage. It is worth noting that, compared with B2 and B3, B1 takes an even shorter matching interval (almost 1s) due to its very stable observation which lacks the benefit of delayed matching.

In general, the intuition behind the learned policy is consistent with the previous analysis in Section 3.2. This implies that incorporating a queueing-based model can intrinsically capture the delayed matching trade-off in the real-world ride-hailing services and adaptively delay the matching to reduce passengers' average wait costs. Integrating a structured model into a model-free RL framework is promising in improving purely data-driven approaches.

6. Conclusion

This paper presents a family of policy gradient RL algorithms for the delayed matching problem. Leveraging the sequential information collected by alternating the matching time intervals when assigning trip requests to idle drivers, the proposed RL algorithm balances the wait time penalties and improved matching efficiency. With prior information on the stationary demand and supply distributions, we first characterize the matching delay trade-off in a double-ended queue model. A multi-server queue model is then used to build a ride-hailing simulator as a model-training environment, weighing the informativeness against dimensionality. Searching for the optimal delayed matching policy in this environment is formulated as a POMDP. We also propose a dedicated stepwise reward decomposition to address the issue of the sparse reward. Finally, we unfold the learned policy into episodic trajectories to garner insights.

We compare the proposed RL algorithm's performance with a baseline strategy using fixed matching time interval and prior work based on real-world taxi data. Our numerical results are twofold. First, we identify the range of supply-to-demand ratio in which the learned delayed matching policies outperform the instantaneous matching strategy. Second, the learned delayed matching policy is both efficient and interpretable. Our results show that the ACER algorithm can reduce the mean total wait time by amount on a par with the baseline. More importantly, it can lower the wait time by 20.41% compared with the instantaneous matching. The learned delayed matching has physical implications. A ride-hailing platform can incorporate the structure of learned strategy as a lookup table to adaptively decide when to use a delayed matching policy and how long these matching delays should be.

This paper leaves several extensions for future research. First, it is worth further investigation and development of an analytical model to quantify the trade-off relationship at a certain supply-and-demand ratio. To obtain this threshold, we must understand how the first-stage matching time interval decision connects to the second-stage bipartite matching rewards. Second, we can extend to a meta-model that switches the agent's matching policy between instantaneous and delayed matching and includes different factors such as passengers' utility into the objective functions. Finally, online RL is needed if implementing the algorithm in a ride-hailing system, which is challenging due to the system's stochasticity. A model must be developed to reflect the system's real-time dynamics so that the agents can learn effectively by interacting with it. However, the system's stochasticity may complicate the agent's target to learn, thereby resulting in an unstable RL algorithm with no convergence guarantees.

CRediT authorship contribution statement

Guoyang Qin: Methodology, Writing - original draft, Visualization. **Qi Luo:** Methodology, Writing - original draft. **Yafeng Yin:** Conceptualization, Methodology, Writing - review & editing, Supervision. **Jian Sun:** Data curation, Funding acquisition. **Jieping Ye:** Conceptualization, Funding acquisition.

Acknowledgments

The work described in this paper was partly supported by research grants from the National Science Foundation (CMMI-1854684; CMMI-1904575) and DiDi Chuxing. The first author (G. Qin) is grateful to the China Scholarship Council (CSC) for financially supporting his visiting program at the University of Michigan (No. 201806260144). We also thank Shanghai Qiangsheng Taxi Company for providing the taxi GPS dataset.

Appendix A. Summary of notations

Table A.1
Notation list of variables and parameters.

Notation	Description
t	A discrete time step in $\{0, 1, \cdots, T\}$
$N_{\mathrm{P}}(t)$	Number of unmatched passenger requests at t
$N_{\mathrm{D}}(t)$	Number of idle drivers at t
$S(t)$	State of the ride-hailing environment at t
$A(t)$	Matching decision; conducting a bipartite matching if $A(t) = 1$ and holding if $A(t) = 0$
$p(s^{'}\|s, a)$	Transition probability from s to $s^{'}$ after taking a
$R(t)$	Reward received from the ride-hailing environment at t
$r(s, a, s^{'})$	Wait reward at $s^{'}$ transitioned from s after taking a
$\pi(a\|s)$	Policy function, the probability of taking action a at s
$V_{\pi}(s)$	State value function, the expected return starting from s by following π
γ	Discount factor, $\gamma \in [0, 1]$
$q_{\pi}(s, a)$	Action value function, the expected total reward starting from s by taking a and then following π
$O(t)$	Partial observation at t
$\lambda_{\mathrm{P}}(t)$	Estimated arrival rate of passenger requests at t
$\lambda_{\mathrm{D}}(t)$	Estimated arrival rate of idle drivers at t
$n_{\mathrm{P}}(t_1 : t_2)$	Actual number of passenger requests arriving between t_1 and t_2
$n_{\mathrm{D}}(t_1 : t_2)$	Actual number of idle drivers arriving between t_1 and t_2
$N_{\mathrm{M}}(t)$	Number of matched passenger-driver pairs at t
$V(t)$	Expected total reward received from the ride-hailing environment at t
$R_{\mathrm{m}}(t)$	Reward regarding the matching wait cost at t
$R_{\mathrm{p}}(t)$	Reward regarding the pickup wait cost at t
$R_{\mathrm{w}}(t)$	Reward returned at t, a combination of $R_{\mathrm{m}}(t)$ and $R_{\mathrm{p}}(t)$
$\tau_{\mathrm{p}}(\cdot, \cdot)$	Total pickup wait cost function
c_{m}	Unit matching wait cost
c_{p}	Unit pickup wait cost
$\boldsymbol{\theta}$	Parameters of the policy function or weights of the actor neural network
$\mathbf{w}$	Parameters of the state value function or weights of the critic neural network
$\pi_{\boldsymbol{\theta}}(a\|s)$	Parameterized policy function or actor neural network
$V_{\mathbf{w}}(s)$	Parameterized state value function or critic neural network
$J(\boldsymbol{\theta})$	Policy performance measure
$\alpha^{\boldsymbol{\theta}}$	Learning rate for the actor neural network
$\alpha^{\mathbf{w}}$	Learning rate for the critic neural network
$G(t)$	Sampled total reward at t
$\eta(s)$	Distribution of s
$\mathbb{S}_0, \mathbb{S}_1, \mathbb{S}$	Subarea 0, subarea 1, and the entire area
(x, y)	Grid (x, y) in a partitioned area
$o(x, y, t)$	observation of grid (x, y) at t
$N_{\mathrm{P}}(x, y, t), N_{\mathrm{D}}(x, y, t)$	Number of unmatched passenger requests/idle drivers of grid (x, y) at t
$\lambda_{\mathrm{P}}(x, y, t), \lambda_{\mathrm{D}}(x, y, t)$	Estimated arrival rate of passenger requests/idle drivers of grid (x, y) at t
$R_{\mathrm{P}}(\mathbb{S}, t)$	Reward regarding the pickup wait cost of subarea $\mathbb{S}$ at $t, \mathbb{S} \in \{\mathbb{S}_0, \mathbb{S}_1\}$

References

Al-Abbasi, A.O., Ghosh, A., Aggarwal, V., 2019. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. IEEE Trans. Intell. Transp. Syst. 20, 4714–4727.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., Rus, D., 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proc. Nat. Acad. Sci. 114, 462–467.

Aouad, A., Saritaç, Ö., 2020. Dynamic stochastic matching under limited time. In: Proceedings of the 21st ACM Conference on Economics and Computation, pp. 789–790.

Ashlagi, I., Burq, M., Dutta, C., Jaillet, P., Saberi, A., Sholley, C., 2018. Maximum weight online matching with deadlines. arXiv preprint arXiv:1808.03526.

Azar, Y., Ganesh, A., Ge, R., Panigrahi, D., 2017. Online service with delay. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 551–563.

Cao, P., He, S., Huang, J., Liu, Y., 2020. To pool or not to pool: Queueing design for large-scale service systems. Oper. Res.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P.H., Kohli, P., Whiteson, S., 2017. Stabilising experience replay for deep multi-agent reinforcement learning. In: International conference on machine learning. PMLR, pp. 1146–1155.
Guo, S., Liu, Y., Xu, K., Chiu, D.M., 2017. Understanding passenger reaction to dynamic prices in ride-on-demand service. In: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE, pp. 42–45.
Hare, J., 2019. Dealing with sparse rewards in reinforcement learning arXiv preprint arXiv: 1910.09281.
Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., 2018. Stable baselines. https://github.com/hill-a/stable-baselines.
Jaakkola, T., Singh, S.P., Jordan, M.I., 1995. Reinforcement learning algorithm for partially observable markov decision problems. In: Advances in neural information processing systems, pp. 345–352.
Ke, J., Yang, H., Ye, J., 2020. Learning to delay in ride-sourcing systems: a multi-agent deep reinforcement learning framework. IEEE Trans. Knowl. Data Eng.
Kim, D.S., Smith, R.L., 1995. An exact aggregation/disaggregation algorithm for large scale markov chains. Naval Res. Logist. (NRL) 42, 1115–1128.
Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., Ye, J., 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In: The World Wide Web Conference, pp. 983–994.
Lin, K., Zhao, R., Xu, Z., Zhou, J., 2018. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1774–1783.
Luo, Q., Huang, Z., Lam, H., 2019. Dynamic congestion pricing for ridesourcing traffic: a simulation optimization approach. In: 2019 Winter Simulation Conference (WSC). IEEE, pp. 2868–2869.
Mao, C., Liu, Y., Shen, Z.J.M., 2020. Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach. Transp. Res. Part C: Emerg. Technol. 115, 102626.
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning arXiv preprint arXiv:1312.5602.
Özkan, E., Ward, A.R., 2020. Dynamic matching for real-time ride sharing. Stochastic Syst. 10, 29–70.
Pardo, F., Tavakoli, A., Levdik, V., Kormushev, P., 2018. Time limits in reinforcement learning. In: International Conference on Machine Learning, pp. 4045–4054.
Qin, G., Li, T., Yu, B., Wang, Y., Huang, Z., Sun, J., 2017. Mining factors affecting taxi drivers' incomes using gps trajectories. Transp. Res. Part C: Emerg. Technol. 79, 103–118.
Sutton, R.S., Barto, A.G., 2018. Reinforcement learning: An introduction. MIT Press.
Wang, H., Yang, H., 2019. Ridesourcing systems: A framework and review. Transp. Res. Part B: Methodol. 129, 122–155.
Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N., 2016. Sample efficient actor-critic with experience replay. arXiv preprint arXiv: 1611.01224.
Williamson, D.P., Shmoys, D.B., 2011. The design of approximation algorithms. Cambridge University Press.
Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., Ye, J., 2018. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 905–913.
Xu, Z., Yin, Y., Chao, X., Zhu, H., Ye, J., 2020a. A generalized fluid model of ride-hailing systems. Working Paper. University of Michigan.
Xu, Z., Yin, Y., Ye, J., 2020b. On the supply curve of ride-hailing systems. Transp. Res. Part B: Methodol. 132, 29–43.
Yan, C., Zhu, H., Korolko, N., Woodard, D., 2020. Dynamic pricing and matching in ride-hailing platforms. Naval Res. Logist. (NRL) 67, 705–724.
Yang, H., Qin, X., Ke, J., Ye, J., 2020. Optimizing matching time interval and matching radius in on-demand ride-sourcing markets. Transp. Res. Part B: Methodol. 131, 84–105.
Zha, L., Yin, Y., Yang, H., 2016. Economic analysis of ride-sourcing markets. Transp. Res. Part C: Emerg. Technol. 71, 249–266.