NondBREM: Nondeterministic Offline Reinforcement Learning for Large-Scale Order Dispatching

Hongbo Zhang[1*], Guang Wang[2*], Xu Wang[1], Zhengyang Zhou[1], Chen Zhang[1], Zheng Dong[3], Yang Wang[1†]

[1]University of Science and Technology of China
[2]Florida State University
[3]Wayne State University
zhanghongbo@mail.ustc.edu.cn, guang@cs.fsu.edu, wx309@mail.ustc.edu.cn, zzy0929@ustc.edu.cn, zhangchenzc@mail.ustc.edu.cn, dong@wayne.edu, angyan@ustc.edu.cn

Abstract

One of the most important tasks in ride-hailing is order dispatching, i.e., assigning unserved orders to available drivers. Recent order dispatching has achieved a significant improvement due to the advance of reinforcement learning, which has been approved to be able to effectively address sequential decision-making problems like order dispatching. However, most existing reinforcement learning methods require agents to learn the optimal policy by interacting with environments online, which is challenging or impractical for real-world deployment due to high costs or safety concerns. For example, due to the spatiotemporally unbalanced supply and demand, online reinforcement learning-based order dispatching may significantly impact the revenue of the ride-hailing platform and passenger experience during the policy learning period. Hence, in this work, we develop an offline deep reinforcement learning framework called NondBREM for large-scale order dispatching, which learns policy from only the accumulated logged data to avoid costly and unsafe interactions with the environment. In NondBREM, a Nondeterministic Batch-Constrained Q-learning (NondBCQ) module is developed to reduce the algorithm extrapolation error and a Random Ensemble Mixture (REM) module that integrates multiple value networks with multi-head networks is utilized to improve the model generalization and robustness. Extensive experiments on large-scale real-world ride-hailing datasets show the superiority of our design.

Introduction

Ride-hailing services (e.g., Uber, Lyft, and Didi) have grown significantly in the last decade due to high mobility demand and the rapid development of the mobile Internet. One of the most important tasks in ride-hailing is real-time order dispatching, which directly impacts passenger experience, driver income, platform profit, and also transportation efficiency. Hence, in this paper, we focus on real-time order dispatching for large-scale ride-hailing services.

Due to the significance of order dispatching, it has attracted lots of interest from both industry and academia, and a plethora of works (Zhang et al. 2017; Xu et al. 2018; Tang et al. 2021; Sadeghi Eshkevari et al. 2022; Sun et al. 2022; Jiang et al. 2023; Qin, Zhu, and Ye 2021; Zhou et al. 2019; Xi et al. 2022; Si et al. 2023; Wang et al. 2023a,b)have been conducted to address this problem. Especially, with the recent advance in deep learning and computing power, deep reinforcement learning (DRL) has shown great potential for sequential decision-making problems such as ride-hailing order dispatching. For example, (Sadeghi Eshkevari et al. 2022) proposed a standalone DRL-based dispatching strategy that is equipped with multiple novel mechanisms to ensure robust and efficient on-policy learning and inference while being adaptable for full-scale deployment. (Tang et al. 2021) designed a value function and hierarchical DRL to unify order dispatching and vehicle relocation problems. (Zhou et al. 2019) developed an independent Q-learning method to solve the communication and interaction problems among multiple agents and the vehicle distribution and order distribution are balanced. (Sun et al. 2022) proposed a novel multi-agent DRL framework to help drivers obtain better orders and make repositioning decisions.

Although those DRL methods show great advantages in the order dispatching problem, they require agents to interact with the environment to collect huge data for model training, which causes a huge challenge to put those algorithms into practice since interacting with the environment online may lead to serious adverse consequences including decreasing drivers' income, passengers' ride experience, as well as increasing operational costs and causing dangerous actions. To deal with these issues, offline DRL as a promising technology has been developed. Compared with traditional online or off-policy DRL, offline DRL trains agents through the logged datasets and does not require agents to interact with the environment, thus avoiding adverse consequences in the learning process. Researchers have been trying to develop offline DRL algorithms for different domains including recommendation systems (Swaminathan et al. 2017) (Xiao and Wang 2021) (Deffayet et al. 2023),healthcare (Fatemi et al. 2022) (Tang et al. 2022), and autonomous driving (Shi et al. 2021) (Fang et al. 2022) and have achieved satisfactory performance. A key challenge to developing offline DRL is to obtain large-scale logged real-world data. Fortunately, due to

*These authors contributed equally.
†Prof. Yang Wang is the corresponding author.
Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

the development of mobile devices and communication technologies, large-scale ride-hailing data including order data and vehicle GPS data are recorded for business and security purposes, and the Data for Social Good initiatives also make them available for research. These data provide a good opportunity for us to develop effective offline DRL algorithms for ride-hailing order dispatching.

However, it is still challenging to develop effective offline DRL for ride-hailing order dispatching even with massive logged data due to the following three key challenges: (i) extrapolation error caused by inconsistencies in the distribution of data accessed by policy and the distribution of the logged data. (ii) Dynamic nondeterministic action space. In ride-hailing order dispatching, each possible order-vehicle match is considered as an action, so both those continuous control algorithms (e.g., BCQ (Fujimoto, Meger, and Precup 2019) and DDPG (Lillicrap et al. 2015) Based) and existing offline DRL algorithms with fixed output units (e.g., Discrete BCQ (Fujimoto et al. 2019)) are not applicable to the order dispatching problem. (iii) Real-time guarantee. We need to generate the decisions in a short limited time to satisfy passenger demand under the scenario with large-scale agents and huge actions.

To address the above three challenges, in this work, we develop a nondeterministic offline reinforcement learning framework called NondBREM for large-scale order dispatching, which learns policy from only the accumulated logged data to avoid costly and unsafe interactions with the environment. In NondBREM, we first construct a dataset for offline training based on real-world logged data, and a Nondeterministic Batch-Constrained Q-learning (NondBCQ) module is then developed to address the algorithm extrapolation error by considering the dynamic nondeterministic action space. A Random Ensemble Mixture (REM) (Agarwal, Schuurmans, and Norouzi 2020) module that integrates multiple value networks with multi-head networks is further utilized to improve the model generalization and robustness. The key contributions of our work are summarized as follows:

- We develop a nondeterministic offline DRL framework for real-time large-scale order dispatching called NondBREM, which learns policy from only logged data and improves performance by limiting action sets. It can be easily integrated into existing order dispatching systems.
- In NondBREM, We design a nondeterministic BCQ module, which can help reduce the extrapolation error and deal with dynamic nondeterministic action spaces. A REM module is also developed to improve the generalization and robustness of the model.
- Extensive experiments on real-world large-scale ride-hailing datasets from over 50,000 vehicles and 20 million orders show that our nondeterministic offline DRL algorithm NondBREM using logged data for large-scale order dispatching achieves better performance than traditional DRL methods in both online and offline scenarios, with a 3.76% order response rate improvement.

Problem Statement

Formally, we model the order dispatching problem as a Markov game (MG) (Littman 1994), which is defined by a five-tuple $G = N(S, A, R, P, \gamma)$, where S is a set of states; A is a set of actions; R is the reward function; P is the transition probability function; γ is a discount factor, and N represents the number of agents. The detailed descriptions of each element are given as follows:

- N: We take vehicles that can be dispatched (i.e., vehicles without passengers) in the current time period as agents and N is the number of available vehicles. We divide the city into a set of grids, and vehicles in the same grid share the same state and action set, i.e., they are homogeneous.
- State $s_t \in S$: s_t represents the global state in time period t. s_t^i represents the state of agent i in period t. Formally it is represented as a four-tuple: $s_t^i = (l, n_v, n_o, t)$, where l is the index of the grid that agent i located, n_v denotes the number of available vehicles in the current grid, n_o denotes the number of orders in the queue at the current grid, and t represents the current time period.
- Action $A_t \in A$: In this problem, we take all possible matches of available drivers and orders as the action set. For each agent i, A_t^i represents the action of agent i in period t. An action can be denoted as a five-tuple (s, e, g, d, p), which consists of the index of the origin grid of the order, the index of the destination grid, the generation time period, the estimated time duration and the price of the order.
- State transitions: We assume that state transitions are deterministic, i.e., there are no cancellations and changes to orders during the delivery process. After T time steps, the agent reaches the destination of the order, and its state will be updated and also receives a reward.
- Reward: The reward determines the objective of the optimization. We use 0.1 times the order price as the reward. This linear transformation is conducive to the convergence of neural networks. This has been proven to be effective both in previous works and in our experiments.

For each agent i at period t, $A_t^i \in A$ and r_t^i represent action set and reward function respectively. The state transition occurs after each decision (i.e., action execution). The state s_t^i of the agent at time t will transform to s_{t+1}^i at time $t+1$, and the agent will receive a reward from the environment. Based on the above definitions, the main purpose of each agent is to learn to maximize the cumulative reward $G_{t:T}$ from t to T,

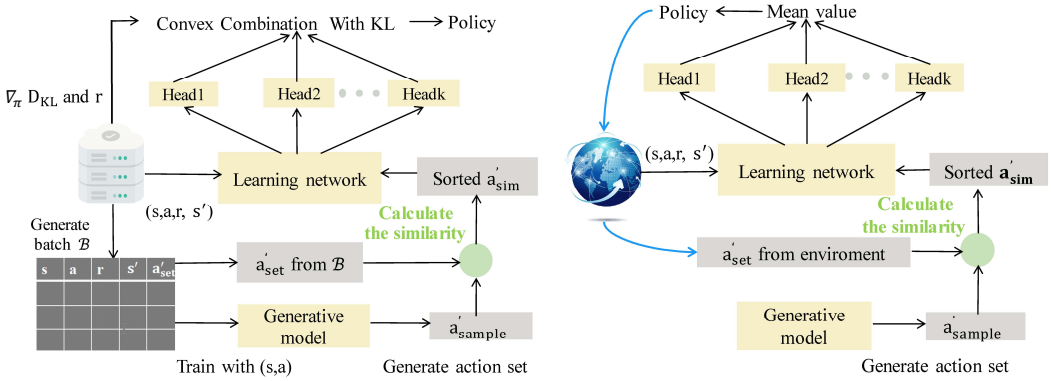
$$\max G_{t:T} = \max \sum_{t=0}^{T} \gamma^t r_t^i, \text{ where } a_t^i \sim \pi_\theta \left(s_t^i\right) \quad (1)$$

where $\pi_\theta(\cdot)$ parameterized with θ represents the policy with respect to the state at time t.

Methodology

NondBREM Framework Overview

In this work, we propose a nondeterministic offline reinforcement learning framework called NondBREM. The



(a) Training process of NondBREM (b) Decision process of NondBREM

Figure 1: (a) describes the training process of NondBREM. The Q-value neural network is divided into two parts, namely the shared learning network of the lower layers and the multi-head network of the upper layers. There are three steps at the green circle in the figure: calculating the similarity matrix of two sets, obtaining the similarity vector from the similarity matrix, sorting and intercepting $a^{'}_{sim}$ according to the similarity to get the set $a^{'}_{sim}$. (b) describes the inference process of the model, which uses a trained network to interact with the simulation environment.

overall framework of NondBREM is shown in Fig. ref-fig:overview. NondBREM consists of two major components: (i) a Nondeterministic Batch-Constrained Q-learning (NondBCQ) module that uses similarity to constrain dynamic nondeterministic action space is developed to reduce the algorithm extrapolation error and complexity, and (ii) a Random Ensemble Mixture (REM) module that integrates multiple value networks with multi-head networks is designed to improve the model generalization and robustness.

NondBCQ

Due to extrapolation errors, traditional online/off-policy RL methods such as DQN and DDPG struggle to adapt to real-world data, so many recent works focus on data-driven DRL (i.e., offline DRL). One of the most popular offline RL methods is Batch-Constrained Q-learning (BCQ), which generates states and actions similar to those in the offline dataset to reduce the distribution offset between the real dataset and generated data accessed by the policy. However, in our setting, each possible match of vehicles and orders is considered an action, so the action set is not fixed and we can only select orders provided by the environment, making BCQ not suitable in our scenario. Addressing the issue of dynamic nondeterministic action space, a NondBCQ method is proposed in this work. In particular, we first remove the disturbance network in the canonical BCQ network. We then calculate the cosine similarity matrix of the generated action set and action set from the real offline dataset, and we finally sort the actions from the real dataset according to the similarity. Therefore, an action set similar to the actions in the dataset is obtained. After that, we utilize the obtained action set to calculate the target q value. Since we have multiple agents in our setting, the environment will become unstable if each agent interacts with the environment, so we optimize the joint policy of agents and maximize the matching rate between vehicle distribution and order distribution as much as possible, leading to the advantage of easily coping with the dynamic agents and lower computational overhead.

Algorithm 1 shows the process of NondBCQ and the details are as follows.

We first get B transitions $(s, a, r, s^{'}, a^{'}_{set})$, where (s, a) is state-action pair at the current time period and $(s^{'}, a^{'}_{set})$ is state-action pair at the next period. $s^{'}$ is used to generate an action set $a^{'}_{sample}$ with the same length n to $a^{'}_{set}$ by a Variational AutoEncoders (VAE) network trained on all (s, a) pairs in a real dataset. A similarity matrix M of $a^{'}_{set}$ and $a^{'}_{sample}$ with size as $n \times n$ is calculated with cosine similarity as measurement. It should be noted that $a^{'}_{set}$ and $a^{'}_{sample}$ are both action set, and the former is the action set from offline data and the latter is the action set generated by the VAE network. The similarity here is utilized to reduce the distribution of actions in the actual decision and the distribution of actions in the offline dataset, thus reducing the extrapolation error. We sort actions in $a^{'}_{set}$ according to the maximum similarity between an action and all actions in $a^{'}_{sample}$, and reserve a part of actions with greater similarity as the new action set $a^{'}_{sim}$. Specifically, the specific operation process of calculating the similarity and constructing the new set is as follows:

(i) We define and calculate a similarity matrix M, where $M[i, j]$ represents the cosine similarity of the i-th element in $a^{'}_{set}$ and the j-th element in $a^{'}_{sample}$.

(ii) After calculating the corresponding similarity matrix, we put the maximum value in each row of the similarity matrix into the similarity vector m, $m[k]$ representing the maximum similarity of the k-th element of $a^{'}_{set}$ to all elements in $a^{'}_{sample}$.

Algorithm 1: NondBCQ

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ,mini-batch size B,Ratio of the maximum candidate order to the number of vehicles β, minimum weighting λ, D_{KL} coefficient η.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$ and VAE $G_\omega = (E_{\omega_1}, D_{\omega_2})$,with random parameters $\theta_1, \theta_2, \omega$,and target networks.$Q_{\theta_1^{'}}, Q_{\theta_2^{'}}$ with $\theta_1^{'} \leftarrow \theta_1, \theta_1^{'} \leftarrow \theta_1$. **Parameter**:$\theta_1, \theta_2, \omega_1, \omega_2$

for $t = 1$ to T **do**
  Sample mini-batch of B transitions (s, a, r, s', a'_{set}) from $\mathcal{B}$
  $\mu, \sigma = E_{\omega_1}(s, a), \tilde{a} = D_{\omega_2}(s, z), z \sim \mathcal{N}(\mu, \sigma)$
  $\omega \leftarrow argmin_\omega \sum(a - \tilde{a})^2 + D_{KL}(\mathcal{N}(\mu, \sigma)||\mathcal{N}(0, 1))$
  Get the length of $a_{set}^{'}$ n and available vehicles in the current area $n^{'}$ from $s^{'}$
  Sample $n_{a_{set}^{'}}$ actions: $\{a_i \sim G_\omega(s')\}_{i=1}^{n}$ as a'_{sample}
  Compute the similarity matrix of a'_{sample} and a'_{set} $M_{i,j}$
  $m[k] = \max(M[i = 1, 2, ...n, j = k])$
  Sort the a'_{set} by vector m in descending order, and truncate the action set size to $\beta \times n$ and get the set $a_{sim}^{'}$
  Compute optimizer goal $\mathcal{L}_\theta$
  $\theta_i \leftarrow argmin\mathcal{L}_\theta$
  Update target networks: $\theta_i^{'} \leftarrow \tau\theta_i + (1 - \tau)\theta_i^{'}$
end for

(iii) We then sort $a_{set}^{'}$ in descending order according to the similarity vector m and crop it to length $\beta * n$ to obtain the action set $a_{sim}^{'}$. The $a_{sim}^{'}$ corresponds to the new action set. In the subsequent calculation of the target q values, the action with the maximal value is selected from the set.

Combined with KL divergence optimization, the optimization objective in Algorithm 1 is as follows:

$$\begin{aligned} \mathcal{L}_\theta &= \Sigma_i(\hat{Q}_{\theta(s,a)} - Q_{\theta_i}(s, a))^2 + \eta D_{KL} \\ \hat{Q}_{\theta(s,a)} &= r + \gamma \max_{a_i \sim a_{sim}^{'}} [\lambda \min_{j=1,2} Q_{\theta_j^{'}}(s', a_i) \\ &\quad + (1 - \lambda) \max_{j=1,2} Q_{\theta_j^{'}}(s', a_i)] \end{aligned} \tag{2}$$

D_{KL} optimization item is used to optimize the distance between order distribution and vehicle distribution after dispatching. This uses a non-explicit communication method to make the movement of multiple vehicles more coordinated. The gradient of D_{KL} to the parameter θ_i is derived using the chain rule:

$$\begin{aligned} \nabla_{\theta_i} D_{KL} &= \nabla_\pi D_{KL} \cdot \nabla_{\theta_i} \pi \\ &= n_t^j \sum_{i=1}^{N} p_{t+1}^i \left[\frac{1}{N_{\text{vehicle}}} - \frac{1}{n_{t+1}^i} \right] \cdot \nabla_{\theta_i} \pi \end{aligned} \tag{3}$$

The gradient of π to θ_i is:

$$\nabla_{Q_i(s,a)} \pi(a \mid s) \cdot \nabla_{\theta_i} Q_i(s, a) \tag{4}$$

N	the number of grids
n_{t+1}^i	the number of idle vehicles in grid i at time $t+1$
n_t^j	the number of idle vehicles in grid j at time t
p_{t+1}^i	the rate of orders in grid i at time $t+1$

Table 1: Important notations

The final gradient of $\mathcal{L}_\theta$ to θ is then calculated as,

$$\nabla_{\theta_i} \mathcal{L}_\theta = \nabla_{\theta_i} \Sigma_i (\hat{Q}_{\theta(s,a)} - Q_{\theta_i}(s, a))^2 + \eta \nabla_{\theta_i} D_{KL} \tag{5}$$

In the calculation of the target q value, the practice of BCQ is followed. Specifically, the convex combination of two q values is used as the target q value. λ has two functions, the first is to give a larger weight to the smaller q value, so as to avoid the overestimation of the q value network. Secondly, the uncertainty of the future time step can be controlled. Here we use the mean square error (MSE) as the loss function. η use to controls the range of D_{KL}. Table 1 illustrates the variables used to calculate the gradient of the KL optimization term. τ is the amplitude of network updating, β is used to control the size of final action set $a_{sim}^{'}$. $n^{'}/n \leq \beta \leq 1$. Where n is the length of $a_{set}^{'}$, which indicates the number of orders in waiting, and $n^{'}$ refers to the vehicles that can be dispatched in the current state. This means that the length of final action set $a_{sim}^{'}$ is between $n^{'}$ and n. When β approaches 1, the action set tends to be uncontrolled. When β approaches $n^{'}/n$, our method selects actions by similarity instead of their q values.

Action Selection Q-learning learns the action value function $Q(s, a)$ with a state-action input. $Q(s, a)$ calculates the expectation of cumulative rewards as,

$$Q(s, a) = \mathbb{E}_\pi[G_{t:T} | s_t^i = s, a_t^i = a] \tag{6}$$

The decision is then made based on the Q function. The Bellman equation of the Q function network can be written as:

$$\begin{aligned} Q\left(s_t^i, a_t^i\right) &= \alpha Q\left(s_t^i, a_t^i\right) + \\ &(1 - \alpha)\left[r_t^i + \gamma \cdot \mathbb{E}_{a_{t+1}^i \sim \pi\left(s_{t+1}^i\right)}\left[Q\left(s_{t+1}^i, a_{t+1}^i\right)\right]\right] \end{aligned} \tag{7}$$

After the policy is obtained through offline training, during inference deployment (i.e., decision) state, we sort actions according to q values and select orders corresponding to the first n maximum Q values as actions. The formula is as follows:

$$\pi(s, n) = a \in \mathcal{A}(s) | Q(s, a) \geq Q(s, a_{n+1}) \tag{8}$$

Where a_i is the action corresponding to the first n maximum q values in state s. n is the number of idle vehicles in the current region. The formula selects orders for all vehicles in the current region.

Random Ensemble Mixture

In order to improve the generalization performance of the algorithm, we utilize Random Ensemble Mixture (REM) to combine multiple q value estimations with a convex combination to a final q value.

Algorithm 2: NondBREM

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ,mini-batch size B,Ratio of the maximum candidate order to the number of vehicles β, loss weighting α, D_{KL} coefficient η.
Initialize multi-head Q-networks $Q_{\theta_1} \dots Q_{\theta_k}$ and VAE $G_\omega = (E_{\omega_1}, D_{\omega_2})$,with random parameters $\theta_1 \dots \theta_k, \omega$,and target networks.$Q_{\theta_1^{'}} \dots Q_{\theta_k^{'}}$ with $\theta^{'} \leftarrow \theta$.
Parameter:$\theta_1 \dots \theta_k, \omega_1, \omega_2$

for $t = 1$ to T **do**
  Sample mini-batch of B transitions (s, a, r, s', a'_{set}) from $\mathcal{B}$
  $\mu, \sigma = E_{\omega_1}(s, a), \tilde{a} = D_{\omega_2}(s, z), z \sim \mathcal{N}(\mu, \sigma)$
  $\omega \leftarrow argmin_\omega \sum(a - \tilde{a})^2 + D_{KL}(\mathcal{N}(\mu, \sigma)||\mathcal{N}(0, 1))$
  Get the length of $a_{set}^{'}$ n and available vehicles in the current area $n^{'}$ from $s^{'}$
  Sample $n_{a'_{set}}$ actions: $\{a_i \sim G_\omega(s')\}_{i=1}^{n}$ as a'_{sample}
  Compute the similarity matrix of a'_{sample} and a'_{set} $M_{i,j}$
  $m[k] = \max(M[i = 1, 2, ...n, j = k])$
  Sort the a'_{set} by vector m in descending order, and truncate the action set size to $\beta \times n$ and get the set $a_{sim}^{'}$
  Compute optimizer goal $\mathcal{L}_\theta$
  $\theta_i \leftarrow argmin\mathcal{L}_\theta$
  Update target networks: $\theta_i^{'} \leftarrow \tau\theta_i + (1 - \tau)\theta_i^{'}$
end for

In the training stage, REM shares the underlying network and uses the output values of multiple upper-layer header networks to calculate the weighted loss function. The weighted loss function is used to train the multi-head and underlying network. The multi-head network can effectively avoid the problem of q estimation bias and enhance the robustness and generalization ability of the policy network.

In the inferring stage, the joint decision is made under the policy constraint. This can make the distribution of data accessed by the policy similar to the logged dataset, thus alleviating the problem of the out-of-distribution query. Meanwhile, the REM algorithm can also enhance the generalization of our model. The pseudo-code for the final NondBREM is shown in Algorithm 2.

The optimization objective $\mathcal{L}_\theta$ in NondBREM is as follows,

$$\mathcal{L}_\theta = \mathbb{E}_{s,a,r,s',a'_{set} \sim \mathcal{B}}[\mathbb{E}_{\alpha \sim \mathrm{P}_\Delta}[\ell_\lambda(\delta_\theta^\alpha)]] + \eta D_{KL} \tag{9}$$

$$\begin{aligned}\delta_\theta^\alpha(s, a, r, s', a_{set}^{'}) = &\sum_k \alpha_k Q_\theta^k(s, a) - r \\ &-\gamma \max_{a' \sim a'_{sim}} \sum_k \alpha_k Q_{\theta'}^k(s', a')\end{aligned} \tag{10}$$

where ℓ_λ is the Huber loss given by,

$$\ell_\lambda(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \le \lambda \\ \lambda\left(|u| - \frac{1}{2}\lambda\right), & \text{otherwise} \end{cases} \tag{11}$$

P_Δ represents a probability distribution over the standard $(K-1)$-simplex $\Delta^{(K-1)} = \{\alpha \in R^K : \alpha_1 + \alpha_2 + ... + \alpha_K = 1; \alpha_k \ge 0; k = 1, \dots K\}$.The gradient of D_{KL} is similar to NondBCQ. η is a hyperparameter used for controlling the range of D_{KL}.

NondBREM trains a family of Q-function approximators defined by mixing probabilities on $(K-1)$-simplex, treating convex combinations of multiple q values as the final q value estimator. We use α_k to weigh the loss of multiple upper-layer networks to obtain the final loss for training multi-headed networks.

In the action selection stage, the average value of K Q-networks is used as the final q value.

$$Q(s, a) = \sum_k Q_\theta^k(s, a)/K \tag{12}$$

Evaluation

Experimental Data

We evaluate our algorithm using real-world ride-hailing data from a large city, over a period of eight weeks. Two types of data are utilized, including vehicle GPS data and more than 20 million order data from over 50K vehicles. The dataset spans from 09/2021 to 11/2021.

- **Vehicle GPS data:** GPS data is collected through onboard equipment on each ride-hailing car. Each GPS record contains fields that describe the real-time status of the vehicle, including vehicle ID, timestamp, and longitude and latitude.
- **Vehicle order data:** The ride-hailing platform will collect information about each order. Features used for dispatching include order generation time, departure latitude and longitude, destination latitude and longitude, order price, order end time, vehicle ID, etc.

Dataset Construction. We use the data to generate a five-tuple dataset $(s, a, r, s^{'}, a_{set}^{'})$ to train the value function. In our setup, the five-tuple data can be divided into four parts: state, action, reward, and number of agents. There are three main types of elements in state and action: location index, time period, number of vehicles, and number of orders. In the GPS data, we have the latitude and longitude of each car, making it easy to get a location index. We divide one day into 144 periods, each of which is 10 minutes long. For rewards, we generate rewards through order prices easily. Finally, GPS data contains rich time and location information, which enables us to obtain the initial idle time and nonworking time of vehicles, thus dynamically controlling the number of vehicles. In summary, we have enough information to generate the data needed to train the offline reinforcement learning algorithm.

It is worth noting that since our simulator uses sampling methods to obtain orders, the order data is used in both the policy training and performance evaluation phases. To evaluate the accurate performance of the model, we made a fixed

division of the data. We use the data from the first 6 weeks for training the model, and the data from the last 2 weeks are loaded into the simulator for performance evaluation.

Baselines and Experimental Setting

The experiments in this section prove that NondBREM can overcome extrapolation errors in offline order dispatching tasks and achieve excellent performance. We evaluated the performance of several algorithms for comparison, and we briefly described each algorithm and the details of the experiments. Several state-of-the-art algorithms are employed as baselines, including,

- **IL (Van Hasselt, Guez, and Silver 2016)**: Each agent uses the same double Deep Q-network for dispatching, regardless of interactions between agents.
- **Tabular Value function (TVal) (Xu et al. 2018):** The state evaluations only consider two variables, i.e., the time and location. The values are obtained by performing dynamic programming in a discrete tabular space, and it updates the tabular with historical data.
- **KL-Based (Zhou et al. 2019)**: Based on the independent order dispatching, the KL divergence optimization term is added to reduce the distance between the available vehicles and the order distribution in waiting, which is an off-policy method.
- **PolarB (Yansheng Wang and Tong 2020)**: A value-based method that received the first prize in the order dispatching task of the KDD Cup 2020 RL track competition.

Noting that KL-Based and PolarB are trained in both online and offline situations and the performance under both settings is reported. We use KL-Based offline and PolarB offline to indicate offline training. The IL method is used as a baseline for training only in online scenarios.

We utilize two widely-adopted metrics to evaluate the proposed method, including the total driver income (also called dispatch score) in a day and the improvement rate of Order Response Rate (ORR) in a day compared to the baseline. Specifically, if the price of an order is denoted as r_a, the dispatch score is defined as

$$dispatch\_score = \sum_{a \in \mathcal{A}} r_a \tag{13}$$

where $\mathcal{A}$ is the set of accepted orders in a day. And the ORR's improvement rate $ORRIR$ of a method M is formulated as,

$$ORRIR_M = (ORR_M - ORR_{IL})/ORR_{IL} \tag{14}$$

The tuned hyperparameters are set as follows. $\gamma = 0.95$, $\tau = 1$, $\lambda = 0.75$, $\beta = min(max(n'/n, 0.9, 1))$, β is selected in a range and we will analyze the influence of different values of β on performance later. Our experiment is implemented in Python with TensorFlow 1.15, and executed under the environment with a CPU as Intel(R) Xeon(R) E5-2620 v4 @ 2.10GHz and one GPU as Nvidia Tesla V100 16GB.

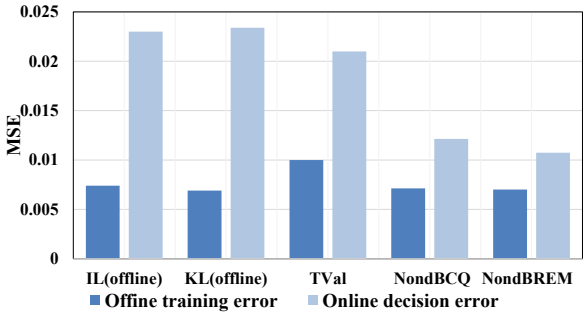


Figure 2: MSE of different algorithms in offline training and online decision-making processes.

Method	Dispatch score	ORRIR(%)
IL	1903297.08	0
TVal	1951672.68	+1.63
KL-Based	2007472.11	+3.12
KL-Based offline	1864928.20	-2.35
PolarB	2024634.79	+2.52
NondBCQ	2005392.15	+2.61
NondBREM	**2043643.57**	**+3.76**

Table 2: Performance under different algorithms. NondBCQ and NondBREM are our proposed algorithms.

Results and Analysis

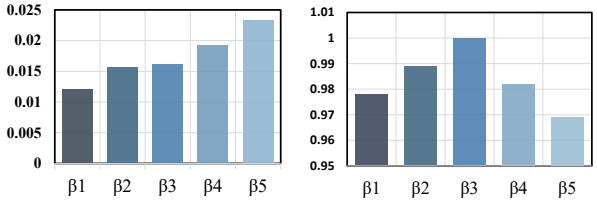
In Table 2, we can see that although traditional reinforcement learning methods have acceptable performance in online scenarios, their performance in offline scenarios fails to outperform that of IL in online scenarios. Our NondBREM uses offline data training, and its performance exceeds that of traditional reinforcement learning algorithms trained online and offline. We report two weeks' average dispatching score and ORRIR.

Meanwhile, in Figure. 2, we report the errors of the five main methods in offline training and online decision-making, errors in the decision stage are only used for evaluation and do not update the Q function. In order to unify the results, we reported mean squared error (MSE). It can be seen that the errors of the five algorithms in the actual decision-making stage are all larger than those in offline training. This error is caused by the inconsistent distribution of training data and policy access data. The error difference between our two methods in the two cases is the smallest, which validates that our method can reduce the extrapolation error and learn an optimal dispatching policy.

Ablation Study

Impact of β. In NondBCQ and NondBREM, the hyperparameter β is used to control the size of the action set, and we have $n^{'}/n \leq \beta \leq 1$. When $\beta = 1$, the size of the action set is the same as the number of orders given by the environment. According to our statistics, n'/n ranges from 0.6 to greater than 1, so we tested five different β values to evaluate the effect of different β values on the algorithm per-

formance, including $\beta_1 = min(max(n'/n, 0.8), 1)$, $\beta_2 = min(max(n'/n, 0.85), 1)$, $\beta_3 = min(max(n'/n, 0.9), 1)$, $\beta_4 = min(max(n'/n, 0.95), 1)$, and $\beta_5 = 1$. We train NondBREM with different values of β and report their losses and normalized cumulative reward. It can be seen in Figure 3 that the cumulative reward value is the largest when $\beta = \beta_3$. However, the loss value is the smallest when $\beta = \beta_1$. When β is small, the action set is also small. Although the corresponding loss is small in this scenario, the actions are too restricted, resulting in poor performance. When β=1, a larger loss leads to a larger error, resulting in poor performance. When β is set appropriately, our algorithm can greatly reduce the extrapolation error and achieve better extrapolation performance.



(a) Mean squared error under different β values

(b) Normalized cumulative rewards under different β values

Figure 3: Loss value and normalized return of NondBREM in online decision process under different values of β.

Effectiveness of the action constraint module. We study the effectiveness of the action constraint module in NondBREM. We change the action constraint module mainly to the extent of VAE-generated network constraints to evaluate its importance. In Figure 3, we can see that when β equals 1, it means we have no restriction on the set of actions, so the algorithm will degenerate into the traditional reinforcement learning method. In this case, the test error of the algorithm becomes large and the cumulative return is small based on the experiment. When a smaller β is selected, the test error becomes smaller and the cumulative return increases. This shows that our action constraint module can effectively reduce the extrapolation error of NondBREM, which is more suitable for offline training.

Methods	25% of Data	50% of Data
	Dispatch score	Dispatch score
KL-Based offline	1831410.23	1834626.25
NondBCQ	1981308.81	1989275.19
NondBREM	2012126.03	2024210.18

Table 3: Performance comparison under different data sizes

Impact of dataset size. We analyzed the influence of different data sizes on the offline algorithms. We experimented with different offline algorithms using 25%, and 50% of the whole data. We extract the corresponding amount of transitions for training. We report the two weeks' average dispatching score in Table 3.

As shown in Table 3, both the traditional algorithms and the offline algorithms achieve better results when more data is available. In experiments with different data sizes, the performance of our algorithm is better than the traditional reinforcement learning algorithm. Moreover, as shown in Figure 4, the larger dataset promotes the convergence of the algorithm. Therefore, in the offline scenario, training on the larger dataset leads to better policy and speeds up the convergence of the algorithm.

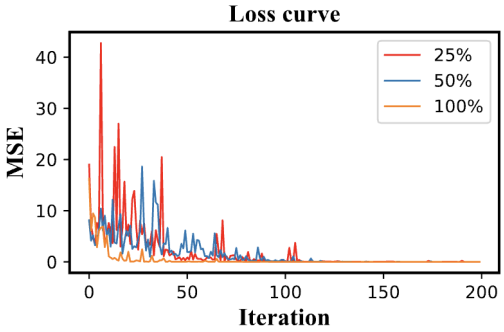


Figure 4: The convergence of NondBREM under different dataset sizes. Each iteration corresponds to 500 training steps and the learning curves for 200 iterations are reported.

Inference time. For large-scale order dispatching, the inference time of the model is important. We thus compare the inferring time consumption of our model with the baselines in Table 4. In the experiments with over 50,000 vehicles (agents), although our model has a longer inference time (1.1435 seconds) due to the VAE network compared to existing methods, it is still short enough for large-scale real-time order dispatching.

Methods	Inference time (s)
IL	0.0526
NondBCQ	1.1277
NondBREM	1.1435

Table 4: The average dispatching time in an episode

Conclusion

In this paper, we design a nondeterministic offline reinforcement learning method called NondBREM for large-scale order dispatching problems. NondBREM learns policy from only the accumulated logged data to avoid costly and unsafe interactions with complicated real-world environments. In NondBREM, a nondeterministic Batch- Constrained Q-learning module (i.e., NondBCQ) using similarity to constrain dynamic nondeterministic action space is developed to reduce the algorithm extrapolation error and a REM module that integrates multiple value networks with multi-head networks is utilized to improve the model generalization and robustness. Extensive experiments on large-scale real-world ride-hailing datasets show the superiority of our design, with 3.76% ORR improvement compared to SOTA baselines.

Acknowledgments

This paper is partially supported by the National Natural Science Foundation of China (No.62072427, No.12227901), the Project of Stable Support for Youth Team in Basic Research Field, CAS (No.YSBR005), Academic Leaders Cultivation Program, USTC, the National Key Research and Development Program of China (No.2022YFB4500300), and the Key Research Project of Zhejiang Lab (No.2022PI0AC01).

References

Agarwal, R.; Schuurmans, D.; and Norouzi, M. 2020. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 104–114. PMLR.

Deffayet, R.; Thonet, T.; Renders, J.-M.; and de Rijke, M. 2023. Offline evaluation for reinforcement learning-based recommendation: a critical issue and some alternatives. In *ACM SIGIR Forum*, volume 56, 1–14. ACM New York, NY, USA.

Fang, X.; Zhang, Q.; Gao, Y.; and Zhao, D. 2022. Offline Reinforcement Learning for Autonomous Driving with Real World Driving Data. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, 3417–3422.

Fatemi, M.; Wu, M.; Petch, J.; Nelson, W.; Connolly, S. J.; Benz, A.; Carnicelli, A.; and Ghassemi, M. 2022. Semi-Markov Offline Reinforcement Learning for Healthcare. In *Conference on Health, Inference, and Learning*, 119–137. PMLR.

Fujimoto, S.; Conti, E.; Ghavamzadeh, M.; and Pineau, J. 2019. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, 2052–2062. PMLR.

Jiang, L.; Wang, S.; Guo, B.; Wang, H.; Zhang, D.; and Wang, G. 2023. FairCod: A Fairness-Aware Concurrent Dispatch System for Large-Scale Instant Delivery Services. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, 4229–4238.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In Cohen, W. W.; and Hirsh, H., eds., *Machine Learning Proceedings 1994*, 157–163. San Francisco (CA): Morgan Kaufmann. ISBN 978-1-55860-335-6.

Qin, Z. T.; Zhu, H.; and Ye, J. 2021. Reinforcement learning for ridesharing: A survey. In *2021 IEEE international intelligent transportation systems conference (ITSC)*, 2447–2454. IEEE.

Sadeghi Eshkevari, S.; Tang, X.; Qin, Z.; Mei, J.; Zhang, C.; Meng, Q.; and Xu, J. 2022. Reinforcement Learning in the Wild: Scalable RL Dispatching Algorithm Deployed in Ridehailing Marketplace. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3838–3848.

Shi, T.; Chen, D.; Chen, K.; and Li, Z. 2021. Offline Reinforcement Learning for Autonomous Driving with Safety and Exploration Enhancement. *arXiv preprint arXiv:2110.07067*.

Si, J.; He, F.; Lin, X.; and Tang, X. 2023. Vehicle Dispatching and Routing of On-Demand Intercity Ride-Pooling Services: A Multi-Agent Hierarchical Reinforcement Learning Approach. *arXiv preprint arXiv:2307.06742*.

Sun, J.; Jin, H.; Yang, Z.; Su, L.; and Wang, X. 2022. Optimizing long-term efficiency and fairness in ride-hailing via joint order dispatching and driver repositioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3950–3960.

Swaminathan, A.; Krishnamurthy, A.; Agarwal, A.; Dudik, M.; Langford, J.; Jose, D.; and Zitouni, I. 2017. Off-policy evaluation for slate recommendation. *Advances in Neural Information Processing Systems*, 30.

Tang, S.; Makar, M.; Sjoding, M.; Doshi-Velez, F.; and Wiens, J. 2022. Leveraging factored action spaces for efficient offline reinforcement learning in healthcare. *Advances in Neural Information Processing Systems*, 35: 34272–34286.

Tang, X.; Zhang, F.; Qin, Z.; Wang, Y.; Shi, D.; Song, B.; Tong, Y.; Zhu, H.; and Ye, J. 2021. Value Function is All You Need: A Unified Learning Framework for Ride Hailing Platforms. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 3605–3615.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Wang, B.; Zhang, Y.; Shi, J.; Wang, P.; Wang, X.; Bai, L.; and Wang, Y. 2023a. Knowledge Expansion and Consolidation for Continual Traffic Prediction With Expanding Graphs. *IEEE Transactions on Intelligent Transportation Systems*.

Wang, B.; Zhang, Y.; Wang, X.; Wang, P.; Zhou, Z.; Bai, L.; and Wang, Y. 2023b. Pattern Expansion and Consolidation on Evolving Graphs for Continual Traffic Prediction. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2223–2232.

Xi, J.; Zhu, F.; Ye, P.; Lv, Y.; Tang, H.; and Wang, F.-Y. 2022. Hmdrl: Hierarchical mixed deep reinforcement learning to balance vehicle supply and demand. *IEEE Transactions On Intelligent Transportation Systems*, 23(11): 21861–21872.

Xiao, T.; and Wang, D. 2021. A general offline reinforcement learning framework for interactive recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 4512–4520.

Xu, Z.; Li, Z.; Guan, Q.; Zhang, D.; Li, Q.; Nan, J.; Liu, C.; Bian, W.; and 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*, 905–913.

Yansheng Wang, Y. L. M. P. Y. X., Dingyuan Shi; and Tong, Y. 2020. Learning to Dispatch and Reposition on a Mobility-on-Demand Platform. KDD Cup 2020 (RL track).

Zhang, L.; Hu, T.; Min, Y.; Wu, G.; Zhang, J.; Feng, P.; Gong, P.; and Ye, J. 2017. A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2151–2159.

Zhou, M.; Jin, J.; Zhang, W.; Qin, Z.; Jiao, Y.; Wang, C.; Wu, G.; Yu, Y.; and Ye, J. 2019. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2645–2653.