Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach

Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, Jieping Ye
AI Labs, Didi Chuxing
{xuzhejesse,lizhixinluca,guanqingwen,zhangdingshui,liqiang_i,nanjunxiao,liuchunyang,bianwei,yejieping}@didichuxing.com

ABSTRACT

We present a novel order dispatch algorithm in large-scale on-demand ride-hailing platforms. While traditional order dispatch approaches usually focus on immediate customer satisfaction, the proposed algorithm is designed to provide a more efficient way to optimize resource utilization and user experience in a global and more farsighted view. In particular, we model order dispatch as a large-scale sequential decision-making problem, where the decision of assigning an order to a driver is determined by a centralized algorithm in a coordinated way. The problem is solved in a learning and planning manner: 1) based on historical data, we first summarize demand and supply patterns into a spatiotemporal quantization, each of which indicates the expected value of a driver being in a particular state; 2) a planning step is conducted in real-time, where each driver-order-pair is valued in consideration of both immediate rewards and future gains, and then dispatch is solved using a combinatorial optimizing algorithm. Through extensive offline experiments and online AB tests, the proposed approach delivers remarkable improvement on the platform's efficiency and has been successfully deployed in the production system of Didi Chuxing.

CCS CONCEPTS

• **Applied computing** → **Transportation**; • **Computing methodologies** → *Multi-agent planning*; *Reinforcement learning*;

KEYWORDS

Intelligent Transportation System; Order Dispatch; Reinforcement Learning, Planning; Multi-agent System

ACM Reference Format:
Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, Jieping Ye. 2018. Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3219819.3219824

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219824

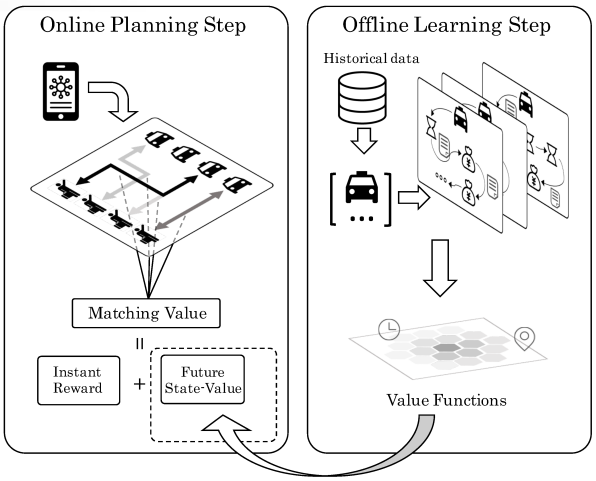


Figure 1: Illustration of the proposed algorithm.

1 INTRODUCTION

We have witnessed a rapid development of on-demand ride-hailing services such as Uber, Lyft and Didi Chuxing in recent years. With the emergence of wireless communication tools, the Global Position System (GPS), and powerful mobile apps, these ride-hailing services provide significant improvements over traditional taxi systems in terms of reducing taxi cruising time and passengers' waiting time [7, 13, 23]. Meanwhile, they also provide rich information on passenger demand and taxi mobility patterns, which can benefit various research areas including demand prediction, route planning, supply chain management, and traffic light control [2, 10, 20, 22].

In this paper, we focus on order dispatch in modern taxi networks [4, 12, 17, 24, 26], which corresponds to the process of finding a proper driver to serve a passenger's request. Previously, greedy methods based on locality are widely used in large taxi companies, such as finding the nearest driver to serve a passenger [8], or using queueing strategies with the principle of first-come-first-serve [25]. Although these methods are easy to implement and manage, they are naturally uncoordinated and tend to prioritize immediate passenger satisfaction over the global supply utilization. Due to the spatiotemporal mismatch between taxi supply and passengers' demand, this could lead to suboptimal results when looking in the long run.

Our goal here is to optimize order dispatch over a long horizon (*e.g.,* several hours or a day), by both fulfilling current passenger

demand and optimizing the anticipated future gain. It relies on a careful modeling of spatiotemporal passenger demand and taxi mobility patterns, as well as quick sensing and controlling in real-time environments. To the best of our knowledge, this is the first work to consider order dispatch with the goal of optimizing long-term global efficiency in a large scale platform such as Didi Chuxing[1].

To accomplish this, we model order dispatch as a sequential decision-making problem. Each individual decision of matching a driver to an order is based on two terms: an instant reward for the driver serving this order attained from real-time information, and an additional term representing the impact of this decision into the future. Based on passenger demand and taxi supply patterns from historical data, we build a unified evaluation metric on spatiotemporal states to quantize the aforementioned future gain (*i.e.,* the learning step). The real-time matching between multiple drivers and orders is formulated as a decision-making problem in multi-agent systems and solved using a combinatorial optimization algorithm that finds the global optimum in a centralized and coordinated way (*i.e.,* the planning step). Figure 1 illustrates the proposed learning and planning algorithm.

We overcome several practical issues to make the proposed algorithm effective in large-scale systems, including computational efficiency, multi-goal optimization, and balancing the tradeoff between user experience and platform efficiency. After extensive experiments on both simulated environments and online AB tests, the proposed method has shown remarkable improvements over baseline algorithms and has been successfully deployed in the production system of Didi Chuxing, serving millions of passengers and drivers in a daily basis.

The contribution of this work is summarized as follows.

- We propose an effective order dispatch algorithm that optimizes long-term platform efficiency for large-scale applications. The algorithm considers both instant passenger satisfaction and the expected future gain in a unified decision-making framework.
- By modeling order dispatch as a sequential decision-making problem with a centralized control, the proposed method naturally falls into the category of reinforcement learning [19]. Implemented in a learning and planning framework, the proposed algorithm is one of the first applications of reinforcement learning in large-scale real-time systems.
- We consider several practical issues such as computational efficiency and experimental design in the proposed algorithm. As a result, the proposed method has been employed in real-world production systems and improves the platform's revenue by a range of 0.5% to 5% in major cities of China.

The rest of the paper is organized as follows. We provide a brief overview of the background and system architecture of order dispatch problem in Section 2. The learning step is detailed in Section 3, including the model definition and the policy evaluation method. Section 4 details the online planning step. Section 5 provides an alternative explanation of the proposed method in reinforcement learning. Experiments are then described in Section 6, followed by a discussion of related works in Section 7. Section 8 concludes the paper.

[1]Didi Chuxing (www.didichuxing.com) serves more than 25 million orders in a day.

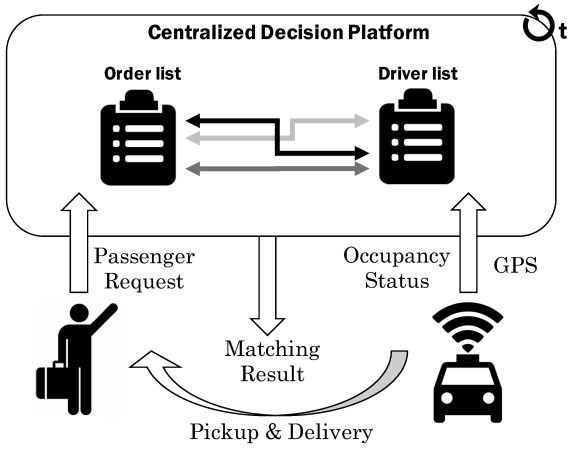


Figure 2: Architecture of order dispatch in on-demand ride-hailing services.

2 ORDER DISPATCH PROBLEM: BACKGROUND AND SYSTEM OVERVIEW

Taxi service is one of the primary transportation services in modern cities. Traditional taxis cruise on streets to discover passengers' requests. In recent years, "on-demand" transportation services like Uber, Lyft and Didi provide a more efficient way of ride hailing. In particular, these online services collect drivers' information and passengers' requests (orders) on the fly and rely on a centralized decision platform to match drivers and orders. As shown by Didi's statistics, the switch from traditional taxis' "driver-select-order mode" to a centralized "platform-assign-order-to-driver mode" leads to a significant improvement on the platform's efficiency, resulting in a more than 10% improvement on the order completion rate.

Order dispatch, which aims to find the best matching between drivers and orders, is obviously crucial for on-demand ride-hailing services [24]. Figure 2 shows a system overview of how it works in practice. Equipped with sensing and communication tools, each taxi periodically uploads its geographical coordinates and occupancy status to the platform. On the other side, when a passenger generates a request "on-demand", he or she will place an order on the platform right away. In the past, Didi relied on a simple but nonetheless successful order dispatch strategy online. In particular, during each short time slot (say one or two seconds), the platform's decision center first collects all the available drivers and active orders, and then matching is based on a combinatorial optimization algorithm.

Given both historical data and real-time taxi monitoring information, we can potentially go beyond optimizing the matching in one time slot. In this paper, we design an order dispatch algorithm that optimizes the platform's global efficiency in a long horizon (*e.g.,* two or three hours or a day). The main idea here is to formulate order dispatch as a large-scale sequential decision-making problem, where each decision corresponds to the action of matching a driver

to an order or not, which maximizes the expected gain of the platform in the long-run. The proposed algorithm differs from previous methods when computing the matching value of a driver-order-pair. Instead of only focusing on the current status (*e.g.*, minimizing pickup distance), we also consider the impact of order dispatch decisions on the future in order to balance the demand and supply distribution spatiotemporally.

3 LEARNING

The learning step aims to provide a quantitative understanding of the spatiotemporal patterns of taxi supply and passenger demand across the whole city. Given historical data, we build a Markov Decision Process (MDP) with an agent representing an individual driver in the platform. The learned value functions produce the "value" for each spatiotemporal state, which is further utilized in the following online planning step.

3.1 Problem Statement

Background. The Markov Decision Process (MDP) is typically used to model sequential decision-making problems. In an MDP, an **agent** behaves in an **environment** according to a **policy** that specifies how the agent selects **action**s at each **state** of the MDP. The agent's goal is to maximize its **gain** $G_t = \sum_{i=t}^{T} R_{t+1}$, *i.e.,* the expected cumulative future rewards starting from time t. To solve an MDP, a common objective is to learn the value functions, including the **state-value function** $V_\pi(s)$ and the **action-value function** $Q_\pi(s,a)$, where $V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$ and $Q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$.

MDP definition. The MDP we build here is in a local view: *each independent driver is modeled as an agent.* Although this setting could lead to a multi-agent problem with thousands of agents for the following planning step, the advantage of this setting is that the definition of state transitions, actions and rewards can be significantly simplified compared to the global-view setting which models the entire platform as an agent. It is worth noting that under the local-view setting, environment contains all the remaining information including order generation and the state of other drivers in the platform. We currently do not distinguish individual drivers - each driver is regarded as the same.

The other components of the MDP are defined as follows and illustrated in Figure 3.

State. The state of the driver is defined as a two-dimensional vector indicating spatiotemporal status. For simplicity, we quantize states into the Cartesian product of a fixed number of time periods and regions, leading to a finite set of states. Formally, we define $s = (t, g) \in S$, where $t \in T$ is the time index and $g \in G$ is the region's index where the driver is located. Note that $|S| = |T| \times |G|$.

Action. There are two main types of actions in our setting. The first type of actions is to assign the driver to serve a particular order. In this case, the agent (driver) will go to the appointed location and pick up the passenger, then send the passenger to the destination, and receive the order's reward.

The other type of actions is idle. This setting is crucial for our problem since drivers may be idle for a long time in particular places (*e.g.* airports and rail stations), making them a less attractive place to go in certain time. Specifically, the idle action means a driver is not matched with any order in one time period. For simplicity, we assume the "idle" action leads to an automatic transition to the next state (same location but in the following time period), with zero reward.

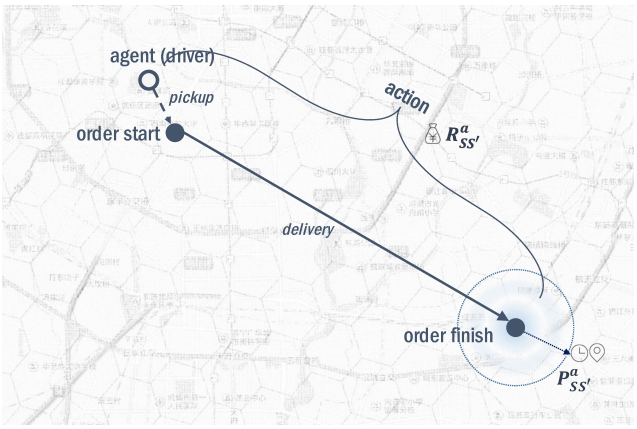


Figure 3: MDP definition in the proposed algorithm. An agent represents a driver; the action in this figure represents assigning the driver to serve a particular order.

Reward. The definition of rewards determines the optimization goal of the whole system. Intuitively, the reward can be defined as the price of an order, by which the goal is to maximize the Gross Merchandise Volume (GMV).

State transition and **reward distribution**. One prominent feature of order dispatch is that the decisions have to be made in a rolling horizon manner. At time t when an order is generated, the system has no exact information for the future, including how much the passenger will need to pay and at what time the trip will complete. As a result, we have to depend on an estimated time of arrival (ETA) and an estimated price when making decisions. The error distribution of these estimations are captured by state transition probabilities $P^a_{ss'}$ and reward distributions $R^a_{ss'}$ implicitly.

Discount factor. The discount factor controls the degree of how far the MDP looks into the future. In our application, it is beneficial to use a small discount factor as long horizons will introduce a large variance on the value function. It is worth noting that under this setting, the reward (order's price) should also be discounted. For an order which lasts for T time slots with price R and a discount factor γ, the final reward is given by

$$R_\gamma = \sum_{t=0}^{T-1} \gamma^t \frac{R}{T}. \tag{1}$$

Let's see an example to make the process clearer. Suppose a driver in area A receives an order from B to C at time 00:00. The trip is estimated to be completed in 20 minutes and costs $30. An additional 10 minutes is needed for the driver to pick up the passenger. Suppose we segment time slots into 10-minute windows (starts at 00:00), and use a discount factor of $\gamma = 0.9$. In our model, this order will make the driver transit from state s to s' with a reward r, where $s = (A, 0)$ and $s' = (B, 3)$ (2 for trip time and 1 for pickup time), and $r = 10 + 10 * 0.9 + 10 * 0.9^2 = 27.1$.

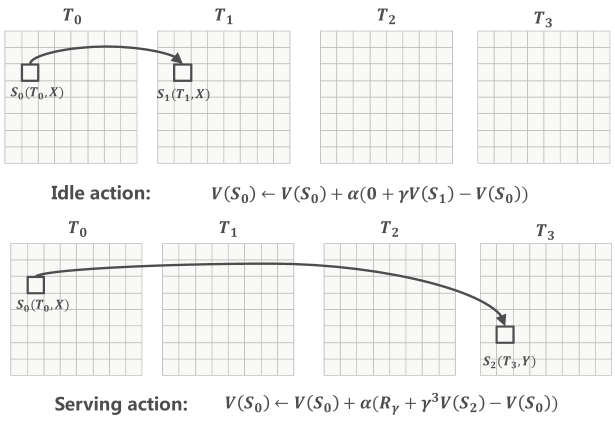


Figure 4: TD update rule. An agent (driver) may undertake two kinds of actions: idle and serving.

3.2 Policy Evaluation

Based on the MDP definition, we break down historical data into a set of transaction pairs (s, a, s', r) including serving actions and idle actions. Note that since we do not distinguish individual drivers, the collected transactions for all drivers can form a single dataset for learning value functions, which corresponds to the concept of policy evaluation.

We assume the online policy that generates transaction data remains unchanged over the training period. Therefore, the subscript π is omitted in this section for short. Specifically, a transaction may come from the action "idle" or the action "serving". As shown in Figure 4 (a), in an idle transaction, the agent gets no immediate reward, and the Temporal-Difference (TD) update rule is as follows:

$$V(s) \leftarrow V(s) + \alpha[0 + \gamma V(s') - V(s)], \tag{2}$$

where $s = (t, g)$ is the current status of the driver, and $s' = (t + 1, g)$ is the following state after s.

On the other hand, in a serving action, the agent receives an immediate reward and triggers a state transition. The TD update rule is computed as:

$$V(s) \leftarrow V(s) + \alpha[R_\gamma + \gamma^{\Delta t} V(s'') - V(s)]. \tag{3}$$

Here $s = (t, g)$ is still the current state of the driver, while $s'' = (t + \Delta t, g_{\text{dest}})$ denotes the estimated finishing state for the driver to complete this order. Specifically, Δt represents the sum of the estimated time of pickup, waiting and deliver process. Note that $\Delta t(\text{idle}) = 1$. g_{dest} records the destination that the passenger provides in the app. The reward should also be discounted using the formula in (1).

The TD update rule is not sample-efficient especially as the number of states may exceed a million in our case. In practice, we perform an implementation based on dynamic programming (DP) to compute the value functions. To make DP possible in our model, we further refine the MDP as a finite-horizon one, where an episode records the transactions of a driver in a day. The DP algorithm is then conducted in an inverted order of time slots, as detailed in Algorithm 1.

Algorithm 1 Policy evaluation (dynamic programming) for the local-view MDP

Input: Collect historical state transitions $D = \{(s_i, a_i, r_i, s_i')\}$; each state is composed of a time and space index: $s_i = (t_i, g_i)$.

Initialize $V(s)$, $N(s)$ as zeros for all possible states.
for $t = T - 1$ to 0 **do**
  Find a subset $D^{(t)}$ where $t_i = t$ in s_i.
  for each sample (s_i, a_i, r_i, s_i') in $D^{(t)}$ **do**
    $N(s_i) \leftarrow N(s_i) + 1$,
    $V(s_i) \leftarrow V(s_i) + \frac{1}{N(s_i)}[\gamma^{\Delta t(a_i)} V(s_i') + R_\gamma(a_i) - V(s_i)]$.
  end for
end for

Return: Value function $V(s)$ for all states

The resultant value function captures spatiotemporal patterns of both the demand side and the supply side. To make it clearer, as a special case, when using no discount and an episode-length of a day, the state-value function in fact corresponds to the expected revenue that this driver will earn on average from the current time until the end of the day.

4 PLANNING

The online planning step takes the learned value functions as inputs and determines the final matching between drivers and orders in real-time. Based on the architecture described in Section 2, we assume that all the supporting information for the matching process, including the geographical coordinates and occupancy status for the drivers, and the origin and destination of orders are given.

4.1 Real-time Order Dispatch Algorithm

In each time slot, the goal of the online order dispatch algorithm is to determine the best matching between drivers and orders. In our problem setting, this goal can also be interpreted as finding the best action for each driver to optimize the future global gain in a coordinated way. Formally, the centralized order dispatch algorithm's objective function is written as:

$$\underset{a_{ij}}{\text{argmax}} \quad \sum_{i=0}^{m} \sum_{j=0}^{n} Q_\pi(i, j) a_{ij} \tag{4}$$

$$s.t. \quad \sum_{i=0}^{m} a_{ij} = 1, \quad j = 1, 2, 3 \ldots, n$$
$$\sum_{j=0}^{n} a_{ij} = 1, \quad i = 1, 2, 3 \ldots, m \quad . \tag{5}$$

where

$$a_{ij} = \begin{cases} 1 & \text{if order j is assigned to driver i} \\ 0 & \text{if order j is not assigned to driver i} \end{cases}$$

Here, $i \in [1, .., m]$ corresponds to all available drivers at this timestamp, while $j \in [1, ..., n]$ corresponds to orders to be served. $Q_\pi(i, j)$ is the action-value function of driver i performing an action of serving order j. Note that the case with $i = 0$ and $j = 0$ corresponds to a special default action which does not serve any order in this timestamp.

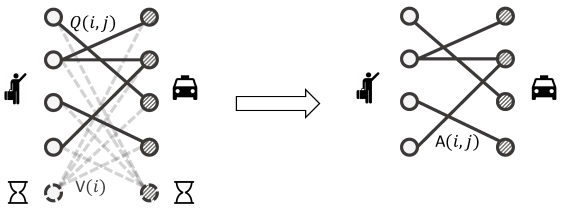


Figure 5: Advantage trick. We remove all connections with the default node.

Note that the agents have a finite action space in that they have to choose one of the available orders to serve or simply doing nothing. On the contrary, in the learning step, we have an infinite action space. The constraints in (5) guarantee that each driver will select one of the available actions including serving orders and doing nothing, while each order could be assigned to at most one driver or being unserved at this timestamp.

We formulate (4) as a bipartite graph matching problem where drivers and orders are two set of nodes; each edge between a driver i and an order j has a weight of $Q_\pi(i, j)$. In practice, we employ the Kuhn-Munkres (KM) algorithm [15] to solve it.

4.2 Advantage Function Trick

With the presence of default actions (drivers doing nothing and orders unserved), the resultant bipartite graph is a complete graph where every possible edge between drivers and orders exists. To reduce computational complexity, we perform an advantage function trick to (4) that eliminates all the edges representing default actions. In particular, since the gap between adjacent rounds of order dispatch (in seconds) is much smaller than the time granularity in the value functions (in minutes), a driver doing nothing in an order dispatch timestamp can be modeled as remaining in the same state. Therefore, the action-value function $Q(i, 0)$ in fact equals to the value function of driver i. This makes it possible to remove all the connections between drivers and the "do nothing" action, by subtracting $V(i)$ to all edges connected to driver i. Figure 5 illustrates the process.

After introducing the advantage function trick, the objective function of order dispatch in one timestamp becomes:

$$\underset{a_{ij}}{\text{argmax}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} A_\pi(i,j)a_{ij} \tag{6}$$

$$s.t. \quad \begin{matrix} \sum_{i=1}^{m} a_{ij} \leq 1, & j = 1, 2, 3 \ldots, n \\ \sum_{j=1}^{n} a_{ij} \leq 1, & i = 1, 2, 3 \ldots, m \end{matrix} . \tag{7}$$

where

$$A_\pi(i,j) = \gamma^{\Delta t_j} V(s'_{ij}) - V(s_i) + R_\gamma(j) \tag{8}$$

is the advantage function. For an edge between driver i and an order j, the advantage function is computed as the expected gain of serving the order (receiving a discounted reward and ending at a state determined by the order's destination) minus the expected value of remaining in the same state (value function of the driver's current state).

The proposed trick eliminates most of the edges in the bipartite graph. As a result, the objective function in (6) can be solved much faster. Algorithm 2 summarizes the planning step.

Algorithm 2 Real-time order dispatch algorithm

Input: Compute value functions $V(s)$ for all states using algorithm 1, and load them as a Look-up-table (LUT).

1: **for** every timestamp of online order dispatch **do**
2: Collect available drivers $i \in [1, ..., m]$ and active orders $j \in [1, ..., n]$.
3: Perform required filters on driver-order pairs and remove invalid connections.
4: Compute advantage function for each valid driver-order pair using (8).
5: Solve (6) using the KM algorithm.
6: Forward the matching information to relevant drivers and passengers. Remaining orders and drivers will go to the next round.
7: **end for**

4.3 Discussion

(6) describes the objective function mathematically. However, it is still unclear how this function interprets in real-life order dispatch systems. To this end, we take a close look at the form of the advantage function in (8). There are four main factors that we will consider:

1) *Order price.* An order j with a higher utility will naturally lead to a higher advantage given the existence of $R_\gamma(j)$.

2) *Driver's location.* The value function of a driver's current state has a negative impact on the advantage function ($-V(s_i)$). Therefore, under the same condition, drivers in areas with lower values are more likely to be selected for serving orders, as the possibility of them receiving other orders in the future is lower than those in higher valued areas.

3) *Order destination.* An action for serving an order whose destination is a more valuable region will possibly achieve higher advantages, as it will lead to a higher $V(s'_{ij})$.

4) *Pickup distance.* Finally, the pickup distance between a driver i and the origin of an order j also contributes to the advantage in an implicit way. Specifically, a larger pickup distance requires more time on picking up the passenger. This will lead to a later time of arrival and increase Δt_j which introduces larger discounts on the destination value function, resulting in lower advantages.

In practice, it is usually needed to deal with multiple objectives. In order dispatch systems, the primary goal is to guarantee good user experience; beyond that, it is preferred to generate more revenue and complete more passenger demands, leading to a win-win situation for drivers, passengers and the platform. This is controlled using a set of hyper-parameters in our system, which are selected based on offline experiments using a simulator.

5 COMBINING LEARNING AND PLANNING

Here we put together what have done in the previous two sections and provide an alternative explanation of the entire process in the perspective of reinforcement learning.

Reinforcement learning aims to find a policy that optimizes the expected future gain for sequential decision-making problems. In our case, the decision center acts as a meta-agent who makes decisions for all drivers (agents) in the platform in a centralized way. It is naturally a sequential decision-making problem, as the meta-agent makes decisions (*i.e.,* find the best matching) in consecutive time slots (say 2 seconds each) and observes the return of total revenue over the next few hours.

The main idea of the proposed algorithm is summarized in the following equations. In each time slot, the meta-learner aims to maximize the expected gain G_t of the global platform:

$$\mathbb{E}_\pi[G_t|S = s], \tag{9}$$

where s is the global state of the whole platform in time slot t.

We decompose the global objective into the product of all individual drivers:

$$\mathbb{E}_\pi[\sum_i G_t^i|S = \{s_i\}], \tag{10}$$

where G_t^i is the cumulative gain of a particular driver i, and s_i is driver i's state.

An approximation is made here that all drivers are independent to each other, so that the global state can be decomposed into the cross product of each driver's state. This is not true in principle, as drivers in the platform have strong relations with each other. However, since drivers can only select actions from a relatively small set of orders in test time and from the fact that most of orders are servable in practice, we can still achieve reasonable results in our application even under this assumption. In that case, the global objective is further decomposed as:

$$\sum_i \mathbb{E}_{\pi_i}[G_t^i|S = s_i]. \tag{11}$$

(11) represents a multi-agent system with each agent having its unique policy π_i. It is still an open question in the literature on how to deal with thousands of self-interest agents in a single model. Therefore, since we depend on a centralized algorithm to make decisions for all agents, we can restrict all agents to have a common-interest of maximizing the global gain, resulting in them sharing the same policy $\pi_i = \pi$. Since maximizing the expected gain equals to maximizing the action-value function, the objective turns to maximize:

$$\sum_i \mathbb{E}_\pi[G_t^i|S_t^i = s_i, A_t^i = a_i] = \sum_i Q_\pi^i(s_i, a_i) \tag{12}$$

Finally, we assume that all drivers are homogeneous so that they can share a set of value functions (*i.e.,* $Q^i(\cdot) = Q(\cdot)$ for all drivers).

$$\sum_i Q_\pi^i(s_i, a_i) = \sum_i Q_\pi(s_i, a_i) \tag{13}$$

Now it is clear that the learning step in Section 3 computes the value functions of an individual driver based on historical data, while the planning step in Section 4 solves (13) in a coordinated way.

It is also possible to incorporate the two steps together and perform an iterative update, which is in fact a policy iteration method: the learning step generates value functions where historical data are sampled using the initial policy; afterwards, the planning step performs a policy update where the goal is to maximize the global gain using the learned value functions.

6 EXPERIMENT

To provide a more complete understanding of the effectiveness of the proposed method, we evaluate it on three environments which correspond to different abstraction levels, including a toy example, a dispatch simulator, and the real-world environment.

6.1 Toy Example

We design a toy example to illustrate the capacity of our MDP framework in handling spatiotemporal order dispatch problem.

Experimental Setup. We consider orders and drivers operating in a simple map of 9×9 spatial grids with 20 time steps. At each time step, drivers are restricted to either stay or move vertically/horizontally by one grid. Meanwhile, orders can only be dispatched to drivers in Manhattan distance that are no greater than 2. An order will be canceled if not being assigned to any driver for a long time, with the cancelation time modeled as a truncated Gaussian in the range 0 to 5 with mean 2.5 and standard deviation 2 along the temporal axis.

For data generation, we want to simulate realistic traffic patterns with a morning-peak and a night-peak, centralized on different locations of residential areas and working areas, respectively. Therefore, orders' starting locations are sampled according to a two-component mixture of Gaussians and then truncated to integers in the spatiotemporal grids. Afterwards, orders' destinations and drivers' initial locations are randomly sampled from a discrete uniform distribution defined on the grids. Parameters of the mixture of Gaussians are as follows.

$$\begin{aligned} \pi^{(1)} &= 1/3, \quad \pi^{(2)} = 2/3; \\ \mu^{(1)} &= [3, 3, 5], \quad \mu^{(2)} = [6, 6, 15]; \\ \sigma^{(1)} &= [2, 2, 3], \quad \sigma^{(2)} = [2, 2, 3]. \end{aligned} \tag{14}$$

Note the three dimensions correspond to the spatial horizontal and vertical coordinates, and the temporal coordinate respectively.

Result Comparison. We compare the proposed method (termed MDP strategy) with two basic strategies - a distance-based dispatch method and a myopic greedy approach. All three variants use KM algorithm on a bipartite graph conducted between drivers and orders. The difference comes from the edge weights - the distance-based method only considers pickup distances between order-driver pairs, while the myopic method only accounts for orders' price as the instant reward. To compute MDP values, we collect transactions from the distance-based method and run the DP algorithm. For comparison measures, we focus on the total revenue defined as the total passenger travel distance for answered orders, together with the average pickup distance and the answer rate (# of answered orders / # of all orders). All the three strategies are tested using 100 orders while the number of drivers being 25, 50 or 75, respectively.

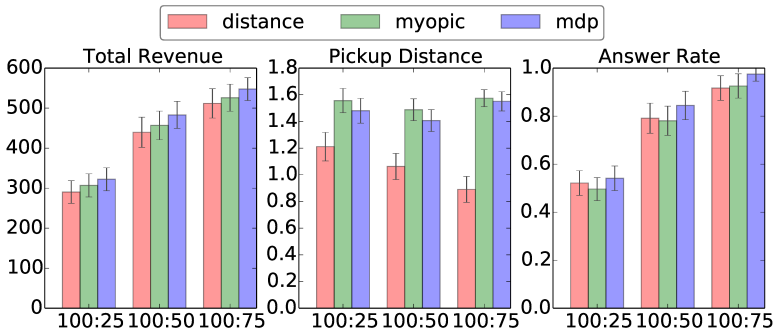


Figure 6: Comparison of distance-based method, myopic method and the proposed MDP method in three metrics on the toy example environment. X-axis stands for the order-driver ratios. Better viewed in color.

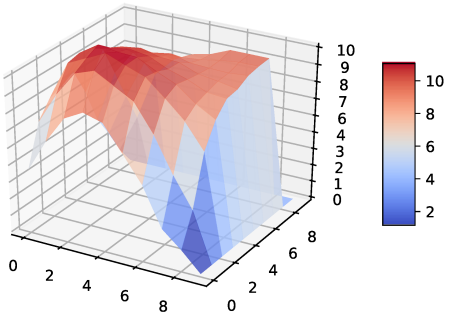


Figure 7: Value function map at time $t = 3$ generated in the toy example environment.

Figure 6 shows comparison results of the three methods for different order-driver ratios, where each bar represents the mean and standard deviation of 1000 independent experiments. It is not astonishing that the distance-based strategy achieves the best pickup distance performance since this strategy prioritizes distance over all other factors. MDP achieves a better revenue and answer rate than both distance-based and myopic strategies. This is consistent with our motivation that a farsighted consideration benefits the overall performance of the dispatch system.

Note that as the driver-order ratio increases, there is a slightly diminishing phenomenon in the performance gain. In other words, the MDP strategy achieves its best performance when there are more demand than supply - this is also not surprising since the method is designed to optimize the global driver utilization and thus is more powerful when the number of drivers is insufficient. Figure 7 illustrates the learned value function at $t = 3$. It is clear that the peak lies close to the first Gaussian's mean area $(3, 3)$.

6.2 Simulator

Beyond the toy example, we further evaluate the proposed MDP method on a more complicated and realistic dispatch simulator. In our application, offline simulator's results are sometimes as important as online experiments considering that the online system is highly dynamic.

Implementation Details. The simulator used here is built to perform a thorough modeling of the physical world for the on-demand ride-hailing platform. A typical sample implementation is to simulate a particular day from historical data. To do so, we first rewind real-happened orders in this date as demands. Each driver is initialized according to the location and time of his or her first entry in the platform. Drivers' actions afterwards are totally determined by the simulator, either according to a user-defined order dispatch algorithm (serving orders), or using models fitted from historical data (idle movements and offline/online operations). The simulator is carefully calibrated, with the difference between simulated results and real-world metrics for the same day being within 2% in most cases, in terms of answer rate and total GMV.

Experimental Results. We implemented the MDP order dispatch algorithm on the simulator and collected results on different cities and dates. Compared to the baseline distance-based strategy, the gain of total GMV ranges from 0.5% to 5% on different cities. Consistent to the discovery in toy examples, the proposed algorithm achieves larger improvements on cities with higher order-driver ratios.

Table 1: Comparison of policy iteration results in the simulator for four median-sized cities. Rate stands for the completion rate for orders.

city	MDP V_0		MDP V_{converge}		
Metric	GMV	Rate	GMV	Rate	Days to Converge
City C	+0.6%	+0.5pp	+0.8%	+0.9pp	4
City D	+0.9%	+1.0pp	+1.4%	+1.5pp	8
City E	-0.1%	+0.1pp	+0.5%	+0.5pp	13
City F	+0.8%	+0.7pp	+1.2%	+1.1pp	7

The improvement on global GMV comes from several changes in the local view. For example, in certain cities, the answer rate for orders heading to high-value areas can be improved by as much as 10% in peak hours using the MDP algorithm. As a result, more drivers could stay or transfer to high-value areas, thus are able to serve more potential orders in the future. Meanwhile, under the same conditions, MDP prioritizes long-trip orders because of their larger instant rewards; we find it a better match with drivers' intention. The simulator is also used to find a proper set of hyper-parameters for online experiments.

Performance & Convergence of Policy Iteration. In Section 5, we mentioned that the proposed learning and planning algorithm can be conducted in an iterative way in the principle of policy iteration. Here we evaluate this approach in the simulator by repeating the same day of simulation as a stable environment and performing policy iteration on it. In particular, we set the distance-based method as the initialized policy and iteratively updates value functions and policy. Table 1 shows the results compared to the baseline distance-based method. The updated policy achieved higher total GMV and completion rate compared to the baseline policy and the MDP policy without policy iteration. This process typically converges within ten iterations in terms of the performance gain.

6.3 Real-World Experiment

Having obtained encouraging results on toy examples and simulated environments, we finally perform real-world experiments on online systems. Before presenting quantitive results, we first detail the design of our experiments.

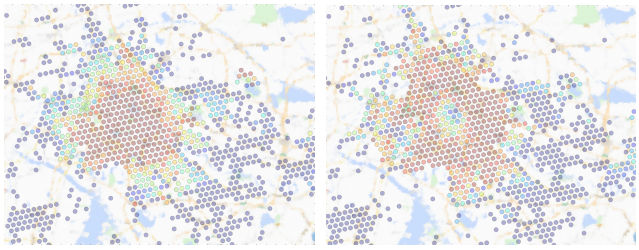
A/B Testing Design. A/B testing is widely-used for performing controlled experiments with two or more variants in real-world systems. Traditionally in web analytics, it is possible to do a simple traffic assigning strategy that splits users intro 50/50 or 90/10 groups for the two compared variants. However, this design does not fit for our problem due to the network effect in matching problems. Imagine if we split drivers into groups, an order may be connected to drivers in multiple groups. It is confusing on how to select a best matching for the order with edge weights computed using different methods.

In practice, we adopted a customized A/B testing design that splits traffic according to large time slices (three or six hours). For example, a three-hour split sets the first three hours in Day 1 to run variant A and the next three hours for variant B. The order is then reversed for Day 2. Such experiments will last for two weeks to eliminate the daily difference. We select large time slices to observe long-term impacts generated by order dispatch approaches.

Experimental Setup and Results. The proposed method has undertaken multiple rounds of A/B testings, covering multiple cities in China. The MDP value functions are computed using dynamic programming based on historical data from the previous month. We compute a separate value for weekdays and weekends. For hyper-parameters, we use a discount factor $\gamma = 0.9$. Again, the proposed MDP method is compared to the baseline distance-based approach. Note that we do not compare to [24] since Didi Chuxing has already completed the switch from the traditional "driver-select-order mode" used in [24] to the new "platform-assign-order mode" studied in this paper, where the switch itself has already brought a significant improvement in efficiency.

Experimental results show that the performance improvement brought by the MDP method is consistent in all cities, with gains in global GMV and completion rate ranging from 0.5% to 5%. Consistent to the previous discoveries, the MDP method achieved its best performance gain in cities with high order-driver ratios. Meanwhile, the averaged dispatch time was nearly identical to the baseline method, indicating little sacrifice in user experience. Given these promising results, the proposed algorithm has been successfully deployed in Didi Chuxing's online dispatch system for more than 20 major cities, serving millions of trips in China in a daily basis.

Visualization of Value Functions. Except for quantitive results, we also notice some interesting phenomenons by visualizing the learned value functions on city maps. For example, Figure 8 shows values across a city in China at two different time slots. Not surprisingly, downtown areas are more valuable than other places in the city. However, by taking a closer look at Figure 8(b), we can observe that the area in the very center place indeed has a lower value at 9:00 when the morning peak just goes by. This phenomenon can be explained from the fact that a mass of drivers have been taken to the center area by orders in the morning peak, so that the place now has far more drivers than requests, resulting in a relatively lower value than nearby areas. A platform-driven guidance is therefore potentially helpful for a more balanced supply distribution at this case. Similar observations have driven many other applications in the platform.



(a) 18:00-18:10, evening peak (b) 09:00-09:10, after morning peak

Figure 8: Sampled value function for the same city at different times. Red indicates higher values, blue for lower ones. Better viewed in color.

7 RELATED WORKS

The design of our MDP dispatch system is closely related to the following two threads of research - taxi dispatch and multi-agent systems.

Taxi Dispatch. In the literature, taxi dispatch usually refers to the process of guiding vacant drivers to find new requests. We term it as *driver dispatch*. For instance, Li *et al.* [7] provided a passenger-finding strategy for driver dispatch according to spatiotemporal features. Qu *et al.* [16] provided another approach that recommends an optimal driving route to maximize drivers' profits. Miao *et al.* [13] presented a receding horizon control approach to reduce driver idle mileage and maintain service quality while handling demand uncertainty. These methods have shown promising results to rebalance the supply side to better serve the demand.

As a special form of taxi dispatch, here we focus on *order dispatch* which assigns a driver to serve an existing order using a centralized decision-making module. Traditional order dispatch methods are typically based on queuing strategies to match supply and demand [6, 17, 25]. Recently, several works on taxi order dispatch integrated various demand prediction [14, 20] components to facilitate dispatch decision making with foresighted considerations. For example, Zhang *et al.* [24] proposed a framework that is capable of serving multiple bookings by combining demand prediction, passengers' destination forecasting and combinatorial optimization. Compared to these researches, our approach provides a more integrated framework for optimizing long-term efficiency for the platform.

Multi-Agent Systems. Multi-agent systems have been extensively studied in a variety of domains including robotic teams, resource management, distributed control, *etc* [1, 3, 9, 21]. In resource management, this problem is investigated by both the community of operation research [18] and reinforcement learning [5, 11]. Compared to solutions from the operation research thread which aim to find a global optimum but usually require large computational overheads, our method has stronger complexity restrictions in order to be implemented in real-time. Thus we model it as a centralized multi-agent problem and solve it using reinforcement learning.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a new order dispatch algorithm that aims to optimize the platform's long-term efficiency, as well as satisfying instant customer demands. To do so, we model order dispatch as a sequential decision-making problem, where the value of each driver-order pair is computed as the sum of an instant reward (the order's utility) and a long-term expected value learned from historical data. Matching between multiple drivers and orders is then determined in a centralized and coordinated way. Experiments on the simulator and online A/B testings reveal the effectiveness of the proposed method, delivering a remarkable improvement in both total revenue and order completion rate compared to the baseline distance-based method. Based on these results, the proposed method has been deployed in the real-world dispatch system in Didi Chuxing.

For future works, we are interested in investigating deep reinforcement learning approaches for order dispatch that directly use raw GPS coefficients as inputs to eliminate boundary effect from region segmentation, and incorporate richer real-time features. There is also a strong motivation in bridging order dispatch and driver dispatch together to balance the supply and demand in a unified framework.

Meanwhile, it is also beneficial to investigate the application of the proposed method in other combinational optimization problems in the domain of logistics, such as reducing the energy usage, supply chain management, and other applications in transportation. We are actively pursuing potential solutions for these tasks.

REFERENCES

[1] Bram Bakker, Shimon Whiteson, Leon Kester, and Frans CA Groen. 2010. Traffic light control by multiagent reinforcement learning systems. In *Interactive Collaborative Information Systems*. Springer, 475–510.

[2] Jie Bao, Tianfu He, Sijie Ruan, Yanhua Li, and Yu Zheng. 2017. Planning Bike Lanes based on Sharing-Bikes' Trajectories. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1377–1386.

[3] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Systems, Man, and Cybernetics, Part C* 38, 2 (2008), 156–172.

[4] Bo Chen and Harry H Cheng. 2010. A review of the applications of agent technology in traffic and transportation systems. *IEEE Transactions on Intelligent Transportation Systems* 11, 2 (2010), 485–497.

[5] Robert H Crites and Andrew G Barto. 1998. Elevator group control using multiple reinforcement learning agents. *Machine learning* 33, 2-3 (1998), 235–262.

[6] Der Horng Lee, Hao Wang, Ruey Long Cheu, Hoon Siew, and Teo. 2004. A Taxi Dispatch System Based on Current Demands and Real-Time Traffic Conditions. *Transportation Research Record Journal of the Transportation Research Board* 1882, 1 (2004).

[7] Bin Li, Daqing Zhang, Lin Sun, Chao Chen, Shijian Li, Guande Qi, and Qiang Yang. 2011. Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. IEEE, 63–68.

[8] Ziqi Liao. 2003. Real-time taxi dispatching using global positioning systems. *Commun. ACM* 46, 5 (2003), 81–83.

[9] Michael L Littman. 2001. Value-function reinforcement learning in Markov games. *Cognitive Systems Research* 2, 1 (2001), 55–66.

[10] Junming Liu, Leilei Sun, Qiao Li, Jingci Ming, Yanchi Liu, and Hui Xiong. 2017. Functional Zone Based Hierarchical Demand Prediction For Bike System Expansion. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 957–966.

[11] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*. 6382–6393.

[12] Michal Maciejewski, Joschka Bischoff, and Kai Nagel. 2016. An assignment-based approach to efficient real-time city-scale taxi dispatching. *IEEE Intelligent Systems* 31, 1 (2016), 68–77.

[13] Fei Miao, Shuo Han, Shan Lin, John A Stankovic, Desheng Zhang, Sirajum Munir, Hua Huang, Tian He, and George J Pappas. 2016. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering* 13, 2 (2016), 463–478.

[14] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. 2013. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems* 14, 3 (2013), 1393–1402.

[15] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[16] Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. 2014. A cost-effective recommender system for taxi drivers. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 45–54.

[17] Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. 2010. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering* 7, 3 (2010), 607–616.

[18] Hugo P Simao, Jeff Day, Abraham P George, Ted Gifford, John Nienow, and Warren B Powell. 2009. An approximate dynamic programming algorithm for large-scale fleet management: A case application. *Transportation Science* 43, 2 (2009), 178–197.

[19] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[20] Yongxin Tong, Yuqiang Chen, Zimu Zhou, Lei Chen, Jie Wang, Qiang Yang, Jieping Ye, and Weifeng Lv. 2017. The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1653–1662.

[21] Michael Wooldridge. 2009. *An introduction to multiagent systems*. John Wiley & Sons.

[22] Carl Yang, Lanxiao Bai, Chao Zhang, Quan Yuan, and Jiawei Han. 2017. Bridging Collaborative Filtering and Semi-Supervised Learning: A Neural Approach for POI Recommendation. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1245–1254.

[23] Daqing Zhang, Lin Sun, Bin Li, Chao Chen, Gang Pan, Shijian Li, and Zhaohui Wu. 2015. Understanding taxi service strategies from taxi GPS traces. *IEEE Transactions on Intelligent Transportation Systems* 16, 1 (2015), 123–135.

[24] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. 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*. ACM, 2151–2159.

[25] Rick Zhang and Marco Pavone. 2016. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. *The International Journal of Robotics Research* 35, 1-3 (2016), 186–203.

[26] Qingnan Zou, Guangtao Xue, Yuan Luo, Jiadi Yu, and Hongzi Zhu. 2013. A novel taxi dispatch system for smart city. In *International Conference on Distributed, Ambient, and Pervasive Interactions*. Springer, 326–335.