Submitted to *Interfaces*
manuscript (Please, provide the manuscript number!)

Ride-hailing Order Dispatching at DiDi via Reinforcement Learning

Zhiwei (Tony) Qin, Xiaocheng Tang, Yan Jiao
DiDi Labs, Mountain View, California,
qinzhiwei@didiglobal.com, xiaochengtang@didiglobal.com, yanjiao@didiglobal.com

Fan Zhang, Zhe Xu, Hongtu Zhu, Jieping Ye
Didi Chuxing, Beijing, China,
feymanzhangfan@didiglobal.com, xuzhejesse@didiglobal.com, zhuhongtu@didiglobal.com, yejieping@didiglobal.com

Order dispatching is instrumental to the marketplace engine of a large-scale ride-hailing platform, such as the DiDi platform, which continuously matches passenger trip requests to drivers at a scale of tens of millions per day. Due to the dynamic and stochastic nature of supply and demand in this context, the ride-hailing order-dispatching problem is challenging to solve for an optimal solution. Added to the complexity are considerations of system response time, reliability, and multiple objectives. In this paper, we describe how our approach to this optimization problem has evolved from a combinatorial optimization approach to one that encompasses a semi-Markov Decision Process model and deep reinforcement learning. We discuss the various practical considerations of our solution development and real-world impact to the business.

Key words: ride-hailing marketplace, order dispatching, reinforcement learning, data-driven decision making

Introduction

With the rising prevalence of smart mobile phones in our daily life, online ride-hailing platforms, that is, mobility-on-demand systems as Alonso-Mora et al. (2017) discuss, have emerged as a viable solution to provide more timely and personalized transportation service, led by such companies as DiDi (www.didiglobal.com), Uber, and Lyft. Urban populations, which are more likely to seek transportation alternatives to car ownership, now account for 59% (National Bureau of Statistics of China 2019) and 82% (University of Michigan Center for Sustainable Studies 2019) of the total population in China and the United States, respectively. These platforms also allow idle vehicle vacancy to be more effectively utilized to meet the growing need for on-demand transportation, by connecting potential mobility requests to eligible drivers. The environment in which trip transactions are facilitated and completed is the marketplace. As Figure 1 illustrates from a passenger's perspective, the marketplace allows a passenger to submit a trip request, the platform to provide a quote, and an available driver to be matched to the trip. The efficiency of the mobility-on-demand marketplace determines how quickly a trip request can be assigned and a passenger can reach his/her destination. It largely hinges on the synergy of supply and demand distributions. If demand exceeds supply locally, passengers would have to wait to receive a response to their requests. Conversely, some drivers would be idle and their utilization would be low. Three major levers are typically used to optimize marketplace operations: order dispatching (matching), driver repositioning (routing), and pricing. Order dispatching and driver repositioning concern the supply distribution, while pricing controls the demand distribution. In this paper, we focus on the optimization of order dispatching for single-passenger trips.

Order dispatching is instrumental in the marketplace engine of a large-scale ride-hailing platform like DiDi, which continuously matches a huge number of trip requests to drivers every day. As Özkan and Ward (2020, p. 31) state, "matching decisions have first-order importance for the ride-sharing firms." We can view order matching as another way of repositioning drivers by using trip orders, and such repositioning movements are fairly deterministic (from the trip origin to the destination). The spatiotemporal distribution of the drivers has a direct impact on the number of trip requests that are matched, the waiting time for the passengers before they are picked up, and eventually the number of orders that can be fulfilled. The efficiency of the marketplace largely hinges on these important factors. Due to the dynamic and stochastic nature of supply and demand in this context, the ride-hailing order-dispatching problem is challenging to solve for an optimal solution. Added to the complexity are considerations of system response time, reliability, and multiple business objectives.

In this paper, we first describe the ride-hailing order-dispatching problem in detail and formulate it in mathematical optimization terms. Then, we describe the evolution of our approach to this

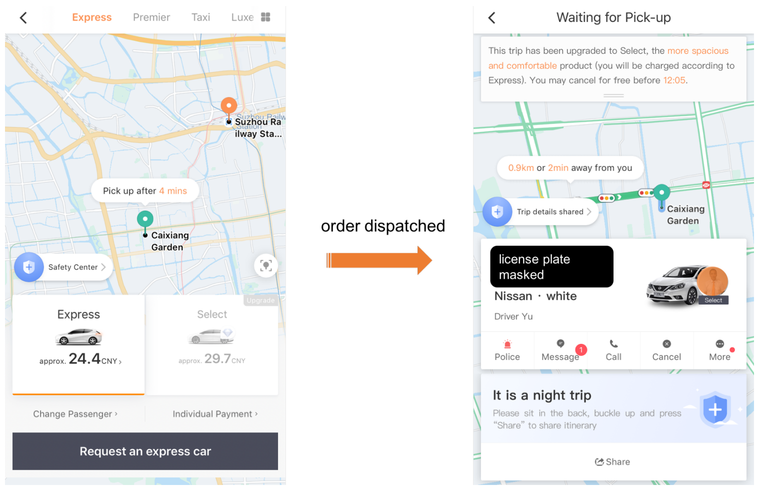


Figure 1 **We Show Sample Screenshots of the Process Involved in Requesting a Ride Using DiDi**

This trip has been upgraded to Select, the more spacious and comfortable product (you will be charged according to Express). You may cancel for free before 12:05.

It is a night trip

Please sit in the back, buckle up and press "Share" to share itinerary

Notes. The screenshot on the left shows a trip inquiry, including the origin and the destination. The screenshot on the right shows a driver matched with a rider, the estimated travel distance, and the wait time. We have masked the driver and vehicle identity information for privacy.

optimization problem from a myopic combinatorial optimization approach to one that encompasses deep reinforcement learning for long-term optimization. Throughout the exposition, we also discuss practical issues arising along the course of our implementation and production experience beginning with the deployment. All the core technical sections have corresponding subsections within the appendix that contain additional mathematical details.

Ride-hailing Order Dispatching (Matching)

In a ride-hailing marketplace, a passenger submits a ride request, including an origin and destination, which the system translates into GPS coordinates. The platform responds with a quote for the trip. The passenger can either proceed to submit the order or cancel it. This is illustrated by the screenshot on the left side of Figure 1. Upon submission, the order enters the dispatching system, which attempts to assign it to an available driver following a particular dispatching policy. The assignment time is typically set by the dispatching policy and cannot be earlier than the order submission time. If no eligible driver is available at that moment, the order waits in the system until the platform is able to match it to a driver. The passenger can cancel the order during this period. If the passenger does not cancel the order, a driver is assigned to the order and can accept or decline the assignment. (Because drivers are motivated by various incentives that DiDi provides, they seldom decline an assignment.) After accepting the assignment, the driver travels from his/her current location to pick up the passenger. The system provides the estimated pickup time and drop-off time to the passenger. (See the screenshot on the right side of Figure 1.) At this time, the passenger can still decide to cancel the order, for example, because the estimated pickup wait time is too long. After the driver picks up the passenger and completes the trip, the passenger pays the charge, and the driver becomes available to be assigned to another order. The driver's income is a predetermined percentage of the actual price. The information about an order is available only when it enters the system. Our problem focuses on finding an optimal online dispatching policy for all orders, considering driver income and passenger waiting time for pickup.

Optimization Problem

Our optimization horizon is 24 hours. A trip order consists of the following information: origin location in the form of GPS coordinates (latitude, longitude), destination location in the same form, order submission time, trip assignment (to the driver) time, pickup time, drop-off time, and price. Note that at dispatching time, the pickup time and drop-off time have to be estimated in conjunction with the candidate driver through a separate module that predicts the estimated time of arrival (ETA) and a module that estimates the charge. A driver is represented by the time that driver last became available and the current spatiotemporal state of the driver. If the driver is in service, then the last available time is set to infinity. An order is eligible to be assigned to a driver if the trip assignment time is later than the last available time of the driver. An order-dispatching policy π is a function that maps an order to a available driver. It is understood that if multiple orders have the same dispatch assignment time, then the dispatching policy ensures that two orders are not matched to the same available driver. If no driver is available, then the order is not matched, and its dispatch request time is advanced to a new decision time when the set of free drivers is no longer empty. We set the price to 0 if the order is cancelled before being fulfilled. A cancellation after an order has been assigned is usually because of a long pickup distance or a long estimated pickup wait time.

We have both driver-centric and passenger-centric objectives. Our driver-centric objective is to maximize the total income of the drivers on the platform. We defer the detailed formulation of the objective in mathematical terms to the appendix. The passenger-centric objective is to minimize the average pickup distance of all the assigned orders. This manages the passenger experience in terms of the waiting time for pickup. The reason for preferring pickup distance over waiting time is that pickup distance is deterministic at assignment time whereas pickup waiting time has to be estimated through a separate ETA prediction module based on a number of factors, such as time, traffic, and weather conditions. We monitor additional marketplace efficiency metrics in terms of response rate and fulfillment rate, which also have an impact on the passenger experience with the platform. Response rate is defined as the percentage of all submitted orders that are assigned to a driver. Fulfillment rate is the percentage of all submitted orders that are eventually completed. Again, the precise mathematical definitions of these metrics are in the appendix. We compare the performance of different algorithms on these metrics in our evaluation process.

Production Requirements and Constraints

Because our solution is targeted for production deployment to match orders and drivers in the real DiDi marketplace, the requirements and constraints of the production system have a significant influence on our choice of a solution approach. For a large-scale dispatching system that serves hundreds of cities, computational efficiency and system reliability are the foremost requirements for any solution. This means that any implementation of neural network inference has to be fast, the service that runs the solution algorithm and is used by the production system has to sustain a high query-per-second (QPS), and the training and serving pipeline has to be of limited complexity. A real-world system needs to be sufficiently flexible to accommodate changing business requirements, some common examples of which are matching eligibility, order priority levels, and multiple objectives. These requirements are frequently rule based and have to be applied as postprocessing functions, if not already explicitly accounted for in the model.

Related Works

The order-dispatching (matching) problem in the ride-hailing domain is related to several classical NP-hard combinatorial optimization problems in the operations research literature. The traveling salesman problem (TSP) dispatches one vehicle to visit multiple known destinations before coming back to its home depot. The vehicle routing problem (VRP) generalizes the TSP to a fleet of vehicles. The order-dispatching problem that we consider in this paper is most closely related to the pickup-and-delivery problem (PDP) or the dial-a-ride problem (DARP), where vehicles are dispatched from and return to a central depot to satisfy a set of transportation requests with single origins and destinations. The ride-hailing order-dispatching problem differs from a DARP in that there is no central depot, and the trip requests are not all known in advance. Hence, the order-dispatching problem is a dynamic problem, in contrast to classical static problems.

Order dispatching has been recognized as an important research topic in ride-sharing applications. A number of works study the optimization problem of dynamic matching (see Özkan and Ward 2020 and the references therein), as opposed to static matching in some early works, for example, nearest-driver matching (Bailey and Clark 1987). Many of them assume time homogeneous driver and order arrival rates. Özkan and Ward (2020) consider time-varying parameters but assume that those arrival rates are given, while Miao et al. (2016) propose a receding horizon control approach. Methods depending on estimates of future demand and supply parameters are generally sensitive to the errors in predictions. Kümmel et al. (2016) and Yan et al. (2019) study order matching with time windows to batch drivers and orders.

Recently, machine learning methods with deep neural networks have been used as new approaches to the TSP and the VRP, under an encoding-decoding framework (Bello et al. 2017, Nazari et al. 2018, Vinyals et al. 2015). Reinforcement learning has been applied to the driver routing problem to increase a driver's prospect of getting an order. Verma et al. (2017) use a Monte Carlo learning approach, and Shou et al. (2020) use dynamic programming to learn the state-action value function. Oda and Joe-Wong (2018) employ a deep Q-network (DQN) in a distributed setting to account for the multiagent nature of the fleet management problem. A Q-network is a state-action value function represented by a neural network.

Solution Approach

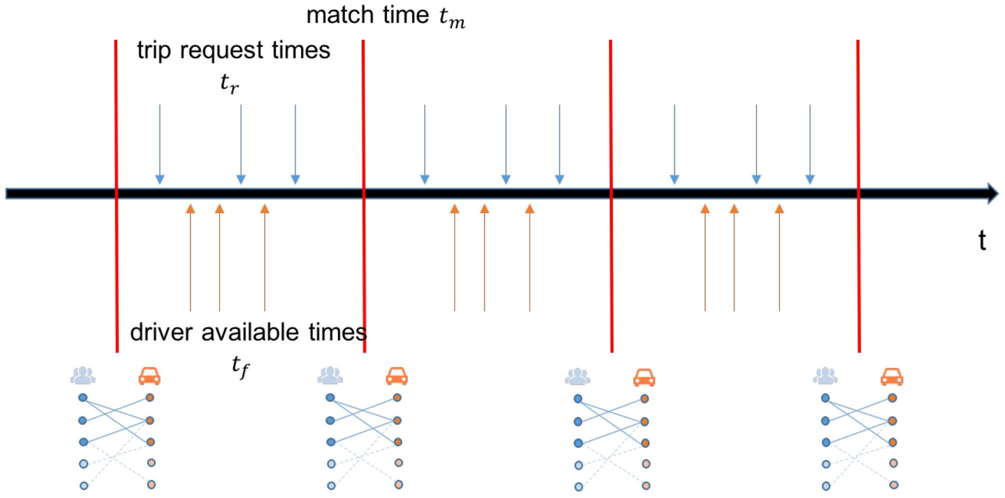
Our two primary objectives are not exclusive. The total income of the drivers on the platform ties closely to the total in-service time (i.e., time during which a driver is serving an order) of the drivers. Of course, because the trip price typically varies depending on the time the trip takes, the relationship between income and in-service time is not linear. Nevertheless, maximizing the total in-service time helps increase the total income of the drivers. Assuming that the number of drivers and their individual (online) available hours are fixed a-priori, maximizing the total in-service time is equivalent to minimizing the sum of pickup wait time and idle time of the drivers, because the total time available to the drivers is fixed. Reducing pickup wait time also decreases the cancellation probability of an order, which has a direct impact on the realization of the trip. We started with a simple combinatorial optimization approach, which sets the foundation of our framework and acts as a policy generator. To develop an approach that optimizes over a longer horizon, we incorporated reinforcement learning into our solution framework by developing tailored algorithms to compute the long-term value of a given dispatching policy, which would in turn guide the policy generator. As we will discuss later, our framework falls under the category of generalized policy iteration, as Sutton and Barto (2018) discuss.

Combinatorial Optimization

Upon an order's arrival, dispatching it to the nearest available driver is a simple common dispatching method (Özkan and Ward 2020, Yan et al. 2019), but it is the most myopic among all the alternatives. A basic step to tackle the stochastic nature of demand and supply in the ride-hailing marketplace is to create dispatching windows, where open orders and available drivers are pooled and matched simultaneously (Kümmel et al. 2016, Yan et al. 2019). Figure 2 illustrates this process. The length of a dispatching window Δt is tunable, typically a few seconds. Batching orders and drivers enables slightly more 'global' optimization at the expense of longer order response time.

Within a dispatching window, there are n open orders and m available drivers denoted by O_{disp} and X_{disp}, respectively. Often, $n > m$, which means that a subset of O_{disp} will not be matched to

Figure 2 We Illustrate the Process of Order Dispatching with Batch Windows



any driver and will have to wait for the next window. Conversely, a subset of X_{disp} drivers will be left idle until the next window. The orders within a batch window are held until the end time of the window for matching. We generate the dispatching policy by solving a linear assignment problem defined by Problem (3) in *Combinatorial Optimization* in the appendix.

Considering various business feasibility constraints, the basic production baseline method has been to set the edge weight to the negative of the pickup distance for the potential match of an order and a driver. In particular, the edge weight is not set to the price, as one might think is intuitive for total income maximization. The reason is that the solution would become to rank O_{disp} in decreasing order of prices and assign any drivers to the top orders when $n > m$, which may result in a longer pickup distance (compared to distance minimization) and short trips that the system ignores repeatedly, thus creating negative customer experiences.

Compared to the nearest-driver matching strategy, combinatorial optimization with batch windows is able to produce a more optimal dispatching policy in terms of pickup distance. However, the batch window length typically has to be small so as not to impact passenger experience. The myopic nature of the solution means that there was still room for further optimization. Özkan and Ward (2020) use an example to illustrate potential efficiency loss due to lack of visibility of the demand in the next batch window.

Although the above combinatorial optimization approach is myopic, it has a number of desirable properties that makes the framework well suited for the production system. First, the optimization algorithm has been well studied in the literature, and fast implementations are available; for example, Lopes et al. (2019) discuss meeting the QPS requirement of a production environment. Second, this method is sufficiently flexible to accommodate various specific business requirements, which we discuss below in the *Production Requirements and Constraints* section by adjusting the edge weights.

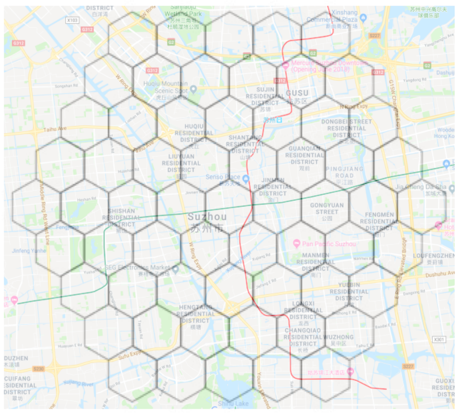
Semi-Markov Decision Process (MDP) Model

To further improve the solution described above, it is desirable to consider the longer-term impact of the current matching decisions, because they affect the availability and distribution of the drivers in the future time steps. The order-dispatching system described in the *Optimization Problem* section is a cooperative multiagent system, with the drivers as the agents. The number of agents poses a big challenge for solving such a multiagent problem because the joint action space quickly becomes intractable. An additional complexity is that the eligible action space (the set of orders to be matched) for each agent changes over time. Holler et al. (2019) demonstrate the challenge of directly learning a global dispatching policy of a system-centric agent. Hence, to develop a practical solution for production, we have taken a driver-centric view of the problem.

The temporal dependency of the dispatching decisions, as discussed in the *Optimization Problem* section above, suggests modeling the dispatching trajectory of each driver as an MDP, which naturally models a sequential decision process that aims to optimize a long-term objective (Xu et al. 2018). In this model, each driver is an independent agent. The *state* s of the driver consists of location and time, both of which can be discretized: the driver's location is represented in a hexagonal grid system as illustrated in Figure 3, and time is represented by buckets, typically of a few minutes. A state is a *terminal* state if its time component is the end time of a day, or an *episode.*

The *action* a of the driver (or rather the action that the system imposes on the driver) is to fulfill a particular order, including to idle or to cruise without a passenger. The *reward* r of an

Figure 3 The Graphic Shows a Sample Map (Obtained from Google Maps) Overlaid with a Hexagon Grid System



action executed on a given state is simply the price of the order, which can be zero if the driver is idle. The state transition dynamics are that after the driver in the current spatiotemporal bucket completes an order, the driver's state changes to the bucket corresponding to the destination, and the driver receives a reward, which equates to a fixed percentage of the price, or equivalently, the amount of the price. An idling or empty-car cruising action is equivalent to an order with a zero price in state transitions. Hence, both the transition and reward are deterministic, given s and a, whereas the action space at s is stochastic. (See the *Semi-MDP Model* in the appendix for more details on this aspect.) The objective of this MDP is to maximize the cumulative reward of the agent (driver) within an episode. It also follows that the objective function of the problem is simply the sum of the objectives of this driver-centric MDP over all the drivers.

This model contains temporally extended courses of actions, so it is in fact a semi-MDP as Tang et al. (2019) describe, and the actions are options (Sutton et al. 1999), which we also denote by o. This is consistent with its meaning because an option is a trip order, with staying idle as a zero-distance trip. See the *The Optimization Problem* in the appendix for the mathematical definition of a trip order. Most of the MDP relevant theories can be carried over with only minor modification (Sutton et al. 1999). The reward r accumulated over an option whose trip portion spans over multiple units of time needs to be properly discounted. Refer to Equation (7) in the appendix for the precise definition. The policy of the agent, π_d is a function that maps the driver's state to an option. We note that in practice, the system executes a centralized dispatching policy π through, for example, solving a linear assignment problem. Nevertheless, π can be distilled into π_d through a dual relationship.

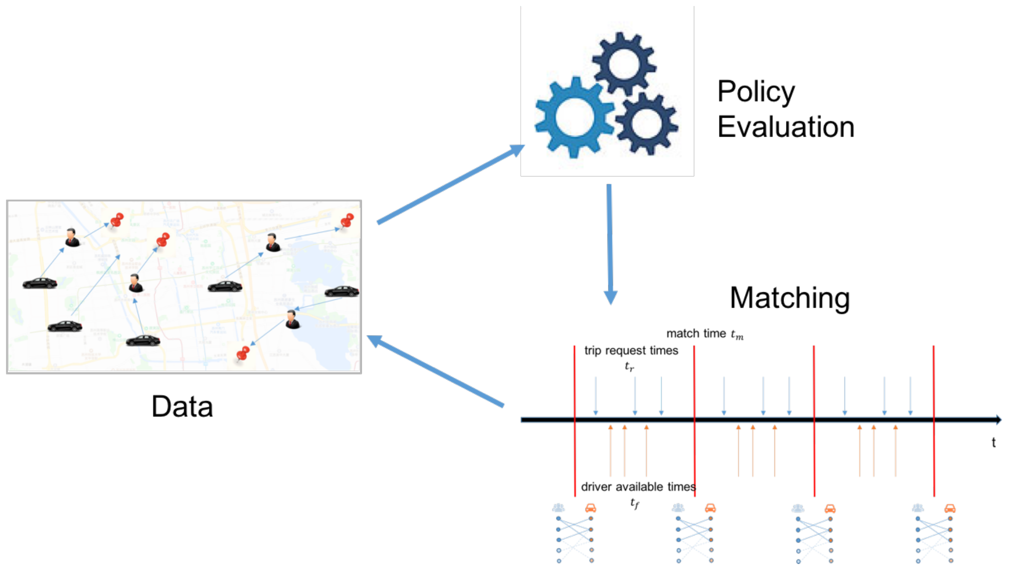
Similar for an MDP, the state-value function $V^{\pi_d}(s)$ of the semi-MDP is defined as the long-term discounted cumulative reward received throughout the options given s, following π_d. Recall that the state space is discretized by the hexagon grid system and time buckets. Hence, $V^{\pi_d}(s)$ can be represented in a tabular form in this case. Similarly, the state-option value function $Q^{\pi_d}(s,o)$ is defined as the long-term discounted cumulative reward received throughout the options by following π_d, given current state s and executing option o on s.

Tabular Temporal-Difference Learning

With the semi-MDP model that we discuss in the previous section, our first reinforcement learning (RL) approach (Xu et al. 2018) employs the generalized policy iteration framework (Sutton and Barto 2018) as illustrated in Figure 4. At the policy evaluation stage, V^{π_d} is learned through tabular temporal-difference (TD) learning (Sutton 1988) (TD(0) to be exact), using the trip and idle movement data for all drivers collected for the training period, for example, a month. We can interpret V^{π_d} as the long-term state value (measured up to the end of the day) of a generic driver at a given location and time. The tabular scheme follows the spatiotemporal grid system defined in the previous section. TD(0) learns the value function by bootstrapping, using the estimate from last iteration to construct the target for update. Algorithm 1 in *Tabular Temporal-Difference Learning* in the appendix lays out the key steps of TD(0), which we applied to the semi-MDP in the previous section. The update term in the inner loop is the *TD-error* for the transition experience, and α is the step size.

The improved system dispatching policy π' with respect to V^{π_d} is generated during the matching stage through the combinatorial optimization discussed above. The edge weights are computed as the sample (predicted) *advantage* (Baird 1993) of each possible match between an order and a driver using V^{π_d}. (See Equation (13) in the appendix.) The driver-centric policy π_d is not explicitly computed and used. Instead, the driver-centric value function is used to generate the system policy

Figure 4 **The Graphic Illustrates the Generalized Policy Iteration Framework for Order Dispatching**



Note. This framework includes policy evaluation (TD-learning and deep value networks), matching (linear assignment with the Kuhn-Munkres algorithm), batch windows of Δt, and data (trips and idle cruising).

π. We emphasize that since the state-value function can be learned offline, the complexity of online serving remains the same as in the case of the myopic distance-based greedy method, that is, solving a linear matching problem.

The advantage can be viewed as the relative change in the long-term value with respect to the current spatiotemporal point of the driver, if the order is assigned to that driver. The generated policy π' is *collective greedy* with respect to V^{π_d} (through the advantage). The sample advantage admits the same form as the TD-error. The edge weight penalizes long pickup distance because increasing estimated en route time decreases the advantage. The new policy is guided by the independent long-term option advantages of the drivers to approximately maximize their total income while discouraging long pickup wait time for the passengers. The frequency of policy update is a tunable variable, which can vary from days to weeks. We formalize our RL framework for dispatch in Algorithm 2 in the appendix. The framework consists of two stages, policy evaluation and policy improvement, as described above.

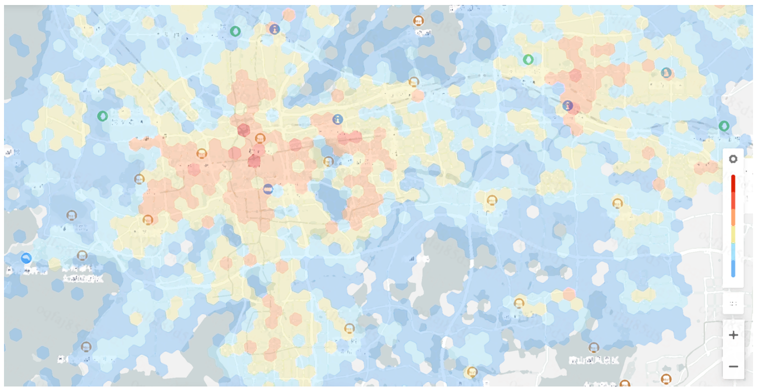
Remark 1. Using the advantages yields the same solution as using the state-option value (which may appear to be a more natural definition of an assignment edge) when every driver is matched to an order within a dispatching window. When the number of available drivers exceeds that of unserved orders within a batch, using an advantage encourages matching a driver whose current state has a lower long-term value than others, while keeping the total value of the solution the same. This is a desirable property because it promotes fairness and helps improve overall driver experiences on the platform, in addition to the ability of penalizing long pickup distances to ensure good passenger experiences.

Deep Reinforcement Learning

Tabular TD-learning has enabled us to improve the system dispatching policy with respect to the average long-term values of individual drivers, but it also has several limitations. (1) Tabular methods suffer from the curse-of-dimensionality. As the number of features to represent the agent state increases, the table size for the value function quickly becomes intractable. (2) The tabular TD method is susceptible to training data sparsity (see Figure 5) because it is unable to generalize in a principled way to spatiotemporal states that have not been visited by any driver in the past. (3) Tabular learning methods do not support the mechanism for knowledge sharing among models of different cities. Meanwhile, using merely spatiotemporal information is not sufficient to capture the complex nature of the driver state. The state space needs to be augmented to enable policies to be more responsive to real-time demand and supply conditions and to better accommodate driver heterogeneity. The dispatching system has to support potentially hundreds of cities with very different data availability. The training method thus needs to be able to leverage knowledge sharing among models of different cities to reduce training time and improve learning quality. With all these considerations in mind, we developed a deep neural network-based policy evaluation algorithm for our generalized policy iteration framework.

The use of a neural network, a nonlinear value function approximator, in value iteration or Q-learning poses known convergence issues (Sutton and Barto 2018). We first demonstrated that a simple neural network model trained within the DQN (Mnih et al. 2015) framework with a set of practical heuristics works effectively for a single-driver dispatching task and transfer learning

Figure 5 The Graphic Shows a Plot of Trip Counts by Hexagonal Grids Over a Day



Note. The darkest blue color indicates single-digit trip counts; zero count is represented by an absence of color.

(Wang et al. 2018). Unlike DQN, which takes only the state as input and has multiple outputs corresponding to each action, our Q-network takes both state and option as input, because the option (trip) space is essentially continuous and is huge if discretized. Transfer learning across cities is facilitated by a new dual-pathway network architecture that distinguishes between transferrable and nontransferrable network components. This architecture is also used later in our new deep value-network (Tang et al. 2019). The key heuristics to ensure successful training include adopting Double DQN (Van Hasselt et al. 2016), sweeping through terminal state transition experiences more frequently from the replay buffer, and augmenting the real trip training data with simulated experience data.

Because the option is part of the input, the Q-network also faces the significant challenge of data sparsity in both state and option spaces. In addition, the matching-trip price and pickup distance influence the network output in a less explicit way than using the state value function. We subsequently developed the cerebellar value-network (CVNet) (Tang et al. 2019) for learning the driver-centric state value function. The development of a CVNet for multidriver order dispatching requires several innovative features in network design and training.

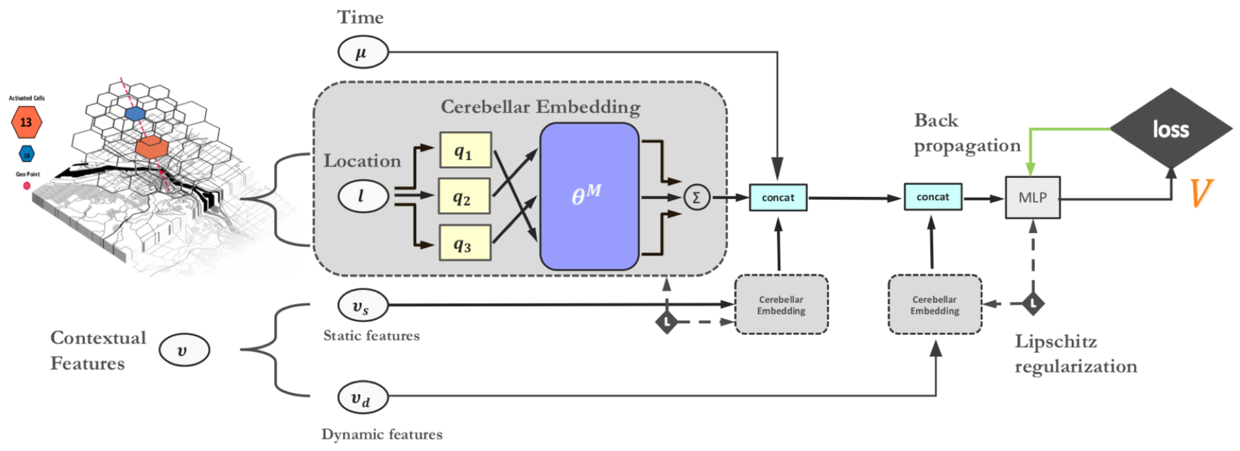
The state values of similar geographical locations at a given time tend to be similar, but due to different densities of the transportation network and natural geographical features, the grouping size of the spatial points can vary. Hence, neither a single-resolution grid system nor the raw GPS coordinates are optimal for state representation and learning. CVNet quantizes the geographical space through hierarchical coarse-coding using a multiresolution hexagonal grid system (Sahr 2011, Brodsky 2018, Uher et al. 2019). State representation is then constructed by combining a Cerebellar Model Arithmetic Computer (CMAC) (Albus 1971) with an embedding matrix. We refer the reader to *Deep Reinforcement Learning* in the appendix for further details on the network architecture of CVNet. The overall network and training architecture is illustrated in Figure 6.

During the matching stage, edge weights are still computed through Equation (13) to generate the dispatching policy. CVNet can be easily distilled into a tabular function form, for example, by sampling origin points from each grid. Hence, CVNet is fully backward compatible with tabular policy evaluation methods within our framework, Algorithm 2. Figure 7 shows two plots of the CVNet output over the grid map of a major Chinese city before morning and evening rush hours. The darker the color, the higher is the value. It is easy to see the different patterns that CVNet captures at these two distinctive times of the day: Before the morning rush hour, the values at suburb areas are higher for an average driver due to the availability of many passengers going to the city center, which subsequently would be an origin of high demand. Before the evening rush hour, the city center has higher values as the trip pattern reverses. The trips during the rush hour largely determine the values of the locations since the time is close to the end of the day (episode).

Transfer Learning

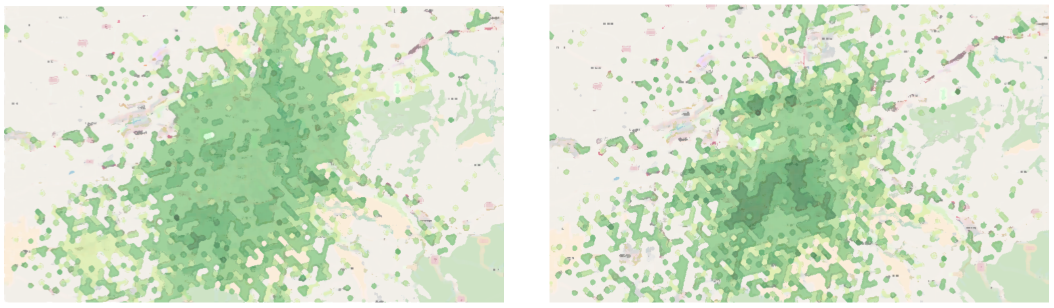
One of the key advantages of employing deep reinforcement learning is to allow leveraging knowledge learned from one city in training to improve training efficiency and quality for other cities. For transfer learning of the models among different cities, we further refined our neural network into a dual-pathway architecture (Wang et al. 2018, Tang et al. 2019), where there are two trains of network layers corresponding to location features (l) and transferrable features such as time (μ), spatiotemporal displacement and local supply-demand contextual features (v). The two pathways are connected thru lateral connections. Figure 8 illustrates this network architecture, which

Figure 6 The Diagram Shows the CVNet Architecture as Tang et al. (2019) Discuss



Note. The left side and center of the diagram illustrate the hierarchical coarse-coding using a multiresolution hexagonal grid system and cerebellar embedding. q_1, q_2, and q_3 represent the quantization functions, and θ^M denotes the parameters of the embedding matrix, defined in *Deep Reinforcement Learning* in the appendix.

Figure 7 We Show Plots of the CVNet Values in a Major Chinese City Before Morning (Left) and Evening (Right) Rush Hours

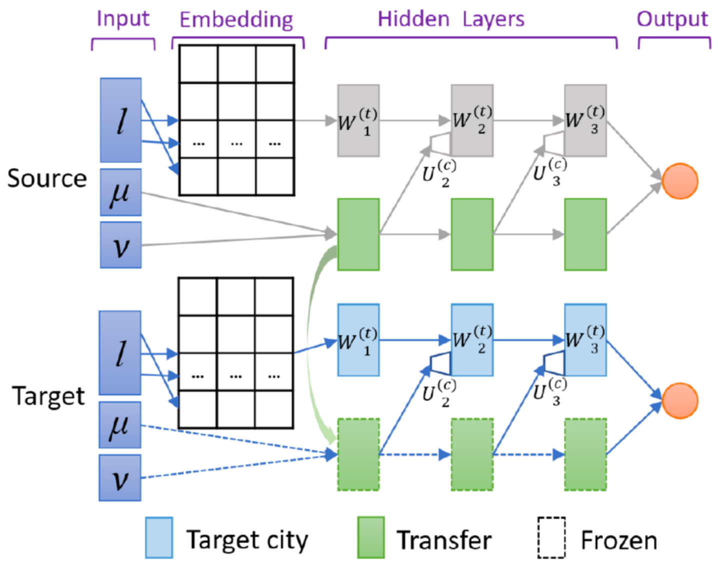


is called the Correlated Feature Progressive Transfer (CFPT) architecture. Once the network for the source city is trained, the transferrable blocks of the network are transplanted to the right positions in the target network, the nontransferrable blocks of which continue to be updated by the new data from the target city.

Performance Evaluation

The primary metrics of interest for evaluation are defined in the *Optimization Problem* subsection in the appendix. We focus on the recount of the empirical evaluation process in this section. For

Figure 8 The Diagram Shows the CFPT Architecture for Transfer Learning (Tang et al. 2019)



Notes. l, μ, and u are the location, time, and contextual features, respectively. The blue blocks in the upper pathway of the target network are the network layers specific to the target city. The green blocks in the lower pathway of both networks are transferrable blocks between the source and target cities. $W^{(t)}$ and $U^{(c)}$ are the network weights of the nontransferrable blocks and the lateral connections, respectively. The parts of the target city network with dotted lines (frozen) are not updated during training.

detailed experiment results, we refer the reader to Xu et al. (2018), Wang et al. (2018), and Tang et al. (2019).

Simulation

Simulation is an integral part of RL research. Typically, agents are trained and evaluated in the same environment, for example, classical toy games (Tsitsiklis and Van Roy 1996), Atari arcade games (Mnih et al. 2015, Schulman et al. 2017), and chess games (Silver et al. 2016), which are all simulations. For industrial applications, the learned policy has to be deployed in real-world production systems, which are usually much more complex and stochastic in system dynamics. We have thus adopted the approach of learning the value function from real trip and trajectory data instead of a simulation environment, and our approach works well with the generalized policy iteration framework. In our setting, we use simulation for evaluating the *relative* performance of a particular policy. Although we have done careful calibrations against real-world data, the ultimate goal of our simulator development is not to exactly replicate the real-world marketplace at the fine-grained level (which is hard to achieve). Instead, it is more important for the simulator to capture the basic dynamics of the interactions of orders, drivers, and the dispatching system.

Our simulation runs by replaying historical passenger orders by days and simulating driver trajectories and behavior with a given dispatching policy. Similar practices have been commonly followed in the literature (Alonso-Mora et al. 2017, Verma et al. 2017). The order replay mechanism allows straightforward construction of an 'out-of-sample' test environment for evaluation. With the simulation environment, we are able to perform hyper-parameter searches, to compare the performance of different algorithm variants within exactly the same supply-demand context, and to demonstrate the effectiveness of transfer learning. Although the supply-demand context can vary significantly for different cities and days, our simulation results show that the RL-based methods consistently outperform the pickup distance minimization baseline with a significant margin. The generalized policy iterations with CVNet achieves the highest total driver income among all the benchmarked methods and exhibits the most robust performance, especially compared with DQN (Tang et al. 2019).

Production

The combinatorial optimization approach has been a core production component that serves order dispatching over all the markets that DiDi covers. It is a high-performance implementation that supports more than 10 billion rides each year from over 550 million passengers, with low latency and high reliability. It also serves as a basis module in our generalized policy iteration framework. The modular design of our approach allows for relatively easy deployment of different solution variants. Both the CVNet and the tabular TD(0) models are trained and updated offline with a Spark-based extract-transform-load (ETL) data pipeline. The output of the trained models are then served online within the combinatorial optimizer to allow it to dispatch orders with consideration of long-term values.

We deployed and tested the reinforcement learning-based solutions in production in a number of pilot cities. Unlike AB tests for many other user-based strategies, which can be done by dividing the experiment's user traffic into two random groups, comparing the performance of two order-dispatching policies is less straightforward and is also an area of research. The key difference is that two dispatching policies cannot be executed at the same time in a given city because it is not possible to completely separate the two groups with a single ride-sharing platform. Cross-group matching of orders and drivers would inevitably happen and the groups would interfere with each other. We adopted the mechanism of *time-slice rotation*. In an AB test of this type, a day is sliced into intervals of H hours, $H \in (0, 24]$. Algorithms A and B are executed on the platform over alternating intervals. The order of the two algorithms are reversed on alternating days. At any time, only one algorithm is running for matching the orders and drivers in the system. In our AB tests, $H = 3$. The number of days within the experiment period is typically chosen to be even. Metrics of interest are collected for each algorithm over the periods that it runs. An advantage of this AB test mechanism is that it allows two dispatching algorithms to run close to parallel while only one algorithm is in control at a time. For methods having a long optimization horizon, a potential problem is that the benefit of Algorithm A's actions may be realized in Algorithm B's intervals. On the other hand, if we set H to be much longer, for example, $H = 24$, then accounting for the effects of test environment changes over different days would be a separate line of research.

Impact

DiDi initially adopted a different mode for order dispatching: Each order was broadcast to all available drivers within a predefined radius. The first driver in this group to accept the order would receive it (Zhang et al. 2017). This mechanism was then replaced by centralized order dispatching with batching, which aims to improve marketplace efficiency by utilizing global supply and demand information. Research on RL-based methods began in 2017. As of this writing, this initiative of using quantitative approaches from operations research and machine learning is in the third year and is still ongoing.

After successful AB tests (with time-slice rotation as we described in the previous section) against the distance-based combinatorial optimization approach, the generalized policy iteration framework with TD(0) has been in production in more than 20 major cities in China (Xu et al. 2018). By design, CVNet tends to have better generalization performance and is less susceptible to data sparsity. Subsequently, CVNet was deployed in production for AB tests in several additional cities and showed significant improvement ($0.5\% - 2\%$) against the production baseline in terms of total driver income, order response rate, and fulfillment rate (Tang et al. 2019). The benefits accrued from the evolution of the order-dispatching algorithm are profound, allowing millions of passengers to have their requests matched faster. The quantitative impact could translate into the equivalence of hundreds of thousand orders per day across the entire market in China. To the best of our knowledge, this is the first major successful application of an RL-based optimization method in the ride-hailing domain.

Works in Progress and Future Directions

We are pursuing a number of research projects to develop and deploy advanced reinforcement learning methods for marketplace optimization, spanning order dispatching, driver repositioning, and carpooling. We are further working on an end-to-end deep RL algorithm to unify dispatching and repositioning (Holler et al. 2019). We briefly discussed the challenges of developing a large-scale multiagent RL approach for order dispatching. Our initial attempt to tackle the problem has yielded some promising results, albeit within a simplified environment (Li et al. 2019). Beyond single-passenger rides, we are also exploring applications of RL in the multiple-passenger (carpooling) setting (Jindal et al. 2018).

Appendix

The Optimization Problem

To facilitate the formulation and statement of the optimization problem, we summarize the quantities in the paper and their notation in Table 1.

Our optimization horizon is 24 hours. A trip order can be summarized as $o := \{\tilde{l}_o, l_o, l_d, t_r, t_m, \tilde{t}_o, t_o, t_d, p\}$. The tilde for $\tilde{l}_o$ and $\tilde{t}_o$ indicates that they have additional dependencies on the assigned driver. We use i to index orders as in $o^{(i)}$. A driver x is represented by $x := \{t_f, l_x, t_x\}$, where t_f is the time when the driver last became available, and (l_x, t_x) is the current spatiotemporal state of the driver. If the driver is in service,

Table 1 We List the Core Mathematical Notation We Use in this Paper

Symbol	Meaning
o	order object
l_o	trip origin in coordinates
l_d	trip destination in coordinates
p	actual trip price
$\hat{p}$	trip price quote
t_r	order submission time
t_m	trip assignment (to the driver) time
$\tilde{t}_o$	driver acceptance (of the assignment of order o) time
$\tilde{l}_o$	driver's location at assignment (of order o) acceptance
$\hat{t}_o$	estimated pickup time
$\hat{t}_d$	estimated drop-off time
t_o	actual pickup time
t_d	actual drop-off time
O_{disp}	set of open orders within a batch window
x	driver object
t_f	time when driver last became available
(l_x, t_x)	current spatiotemporal state of driver
X_{disp}	set of available drivers within a batch window

Note. Notations not listed in this table are explicitly defined and explained in the paper when they are presented.

$t_f = \infty$. The duration of the trip $o^{(i)}$ is ${\tau_o}^{(i)} := {t_d}^{(i)} - {t_o}^{(i)}$. The time that the driver spends en route to pick up the passenger is ${\tau_e}^{(i)} := {t_o}^{(i)} - \tilde{t}_o^{(i)}$. The total time for fulfilling the order is thus ${\tau_o}^{(i)} + {\tau_e}^{(i)} = {t_d}^{(i)} - \tilde{t}_o^{(i)}$. An order o is eligible to be assigned to a driver x if $t_m(o) \geq t_f$, where $t_m(o)$ is the t_m component of o (and this notational convention applies to the other quantities as well). For simplicity, we shorten the notation $t_m(o^{(i)})$ to ${t_m}^{(i)}$. We use j to index drivers as in $x^{(j)}$. We denote the set of free drivers by $X(t) := \{x^{(j)} \mid {t_f}^{(j)} \leq t\}$. An order-dispatching policy π is a function that maps an order o to a free driver $x \in X(t_m(o))$, that is,

$$\pi(o) : o \rightarrow x \in X(t_m(o)). \tag{1}$$

It is understood that if multiple orders have the same dispatch assignment time t_m, then the dispatching policy π ensures that two orders are not matched to the same free driver. If $X(t_r(o)) = \emptyset$, then the order is not matched, and its dispatch request time is advanced to the new decision time until $X(t_r(o))$ is nonempty.

We set the price $p = 0$ if the order is cancelled before being fulfilled. A cancellation after an order has been assigned is usually because of a long pickup distance or a long estimated pickup wait time $\hat{t}_o^{(i)} - \tilde{t}_o^{(i)}$), defined by $d(\tilde{l}_o^{(i)}, {l_o}^{(i)})$, where the function d returns the travel distance between two locations. Hence, p is a function of π, and we can make it explicit by writing $p(\pi)$.

Let the set of orders created by passengers throughout a day be $\{o^{(i)}\}_{i=1}^N$. We have both driver-centric and passenger-centric objectives. Our driver-centric objective is to maximize the total income of the drivers on the platform. By definition, the per-order income of a driver is $r := p\theta$, where θ is a constant independent from the order. Hence, the optimization problem is

$$\max_\pi \; J(\pi) := \sum_{i=1}^N p^{(i)}(\pi). \tag{2}$$

The passenger-centric objective is to minimize the average pickup distance of all the assigned orders, $\frac{1}{N^+}\sum_{i=1}^N (d(\tilde{l}_o^{(i)}, {l_o}^{(i)}))_{\mathbf{1}(\tilde{l}_o^{(i)} \neq \emptyset)}$, where N^+ is the number of assigned orders. This manages the passenger experience in terms of the waiting time for pickup. Response rate is defined as the percentage of all submitted orders that are assigned to a driver, $\frac{N^+}{N}$. Fulfillment rate is the percentage of all submitted orders that are eventually completed, $\frac{\sum_{i=1}^N \mathbf{1}(p^{(i)}(\pi) > 0)}{N}$.

Combinatorial Optimization

Within a dispatching window, we set $t_m(o) = t_{disp}, \forall o \in O_{disp}$, where t_{disp} is the end time of the window. A weighted bipartite graph $\mathcal{G} := \langle O_{disp}, X_{disp} \rangle$ is created with n order nodes and m driver nodes. The edges are pruned first by the dispatch radius; for each order, the edges to those drivers that are farther from a predefined threshold are eliminated. The edge weight w_{ox} determines the objective of interest. The dispatching policy is generated by solving a linear assignment, Problem (3), based on $\mathcal{G}$, using the Kuhn-Munkres (KM) algorithm, also known as the Hungarian algorithm (Kuhn 1955).

$$\begin{aligned} \max_z \; & \sum_{o \in O} \sum_{x \in X} w_{ox} z_{ox} \\ s.t. \; & \sum_x z_{ox} \leq 1, \; \forall o \in O_{disp}, \\ & \sum_o z_{ox} \leq 1, \; \forall x \in X_{disp}, \\ & z_{ox} \in \{0, 1\}, \; \forall o \in O_{disp}, \; x \in X_{disp}. \end{aligned} \tag{3}$$

With the optimal solution z^*, the dispatch policy for the current window can be derived by setting

$$\pi(o) = x, \quad \forall o \in O_{disp},\ x \in X_{disp},\ z^*_{ox} = 1. \tag{4}$$

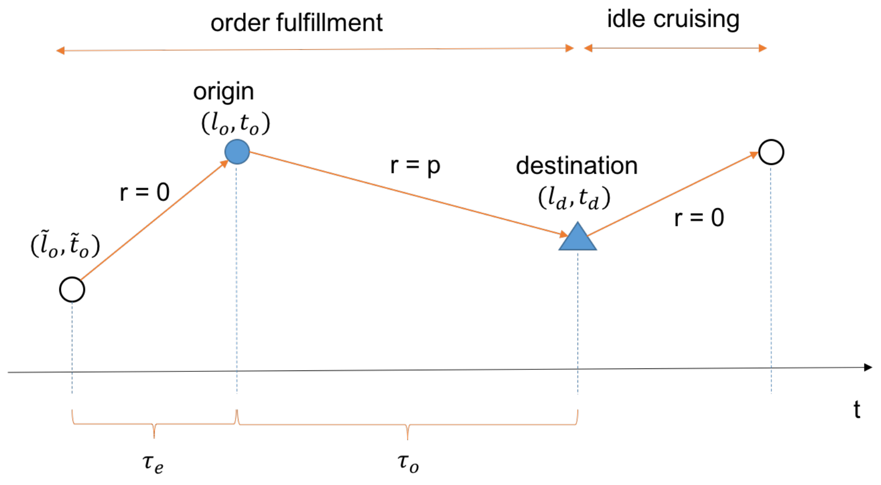
We define an operator $\Pi(w): w \to \pi$ that maps the input weights w to a policy via Equations (3) and (4). Since π is a system-level dispatching policy, it requires not only the order of concern as input, but O_{disp} and X_{disp} as well. For notational simplicity, we keep π in the current form.

Semi-MDP Model

In this model, each driver is an independent agent. The state s of the driver consists of location and time (l, t), both of which can be discretized: the driver's location is represented in a hexagonal grid system as illustrated in Figure 3, and time is represented by buckets, typically of a few minutes. A state is a *terminal* state, if $t = T$ where T is the end time of a day, or an episode. We denote the (hexagonal grid index, time bucket index) pair of location l and time t by $g(l, t)$. The hexagonal grid system is commonly used in mapping systems because it has a desirable property that the Euclidean distance between the center points of every pair of neighboring grid cells is the same, and hexagonal grids have the optimal perimeter/area ratio, which leads to a good approximation of circles (Hales 2001). The action a of the driver (or rather the action that the system imposes on the driver) is to fulfill a particular order (from the open orders within a matching window), or to idle. The reward r of an action executed on a given state is simply the price of the order p, which can be zero if the driver is idle. The state transition dynamics are that after the driver at $s = g(\tilde{l}_o, \tilde{t}_o)$ completes an order o, the driver's state changes to $s' = g(l_d, t_d)$, and the driver receives a reward of $r = p$. Hence, both the transition and reward are deterministic, given s and a. A sample trajectory of the driver in an episode is shown in Figure 9. The stochasticity of this MDP lies in the future demand, which defines the feasible action set at each state. Hence, strictly speaking, our MDP is one with stochastic action sets (SAS-MDP) (Boutilier et al. 2018). Regular learning algorithms like Q-learning, DQN, and policy evaluation methods still work the same in this case using batch data, as long as updates are made over realized available actions. The objective of this MDP is to maximize the cumulative reward of the agent (driver) x within an episode, $J_x = \sum_{k=1}^{K} r_{t_k}$, where t_k is the time of the k-th action, and t_K is the time of the last action before T. For notational simplicity, we subsequently may just use r_k for r_{t_k}. Since all idle actions yield zero reward, we have

$$J_x(\pi) = \sum_{k=1}^{K} r_{t_k} = \sum_{i=1}^{N} p^{(i)}(\pi) \Big|_{\pi(o^{(i)}) = x}. \tag{5}$$

Figure 9 The Graphic Shows a Sample Driver Trajectory



Notes. The first two legs correspond to a transition for order fulfillment (i.e., pickup plus actual trip). The last leg is a transition for idle cruising. The definitions of the notation can be found in *The Optimization Problem* in the appendix.

From Equation (5), it follows that

$$J(\pi) = \sum_x J_x(\pi). \tag{6}$$

This model contains temporally extended courses of actions, so it is in fact a semi-MDP as Tang et al. (2019) describe, and the actions are options (Sutton et al. 1999), which we also denote by o (and is consistent with its meaning). Most relevant theories of MDP can be carried over with only minor modification (Sutton et al. 1999). The reward r accumulated over an option whose trip portion spans over τ_o units of time needs to be properly discounted:

$$\hat{r} = \gamma^{\tau_e}\left(\frac{r}{\tau_o} + \gamma\frac{r}{\tau_o} + \cdots + \gamma^{\tau_o - 1}\frac{r}{\tau_o}\right) = \frac{r(\gamma^{\tau_o} - 1)\gamma^{\tau_e}}{\tau_o(\gamma - 1)}, \tag{7}$$

where $\gamma \in [0,1)$, $\tau_o \geq 1$ and assuming that the reward is accumulated uniformly over time. Here, we have also taken into account the time that the driver spent en route to pick up the passenger, τ_e. The policy of the agent π_d is a function that maps the driver's state to an option, that is, $\pi_d(s): S \to O$, where S and O are the state and option spaces, respectively. Note that the system dispatching policy π has a different order of input and output, because it is a system-view policy and it is more notationally convenient to define it that way. Nevertheless, π can be distilled into π_d through

$$\pi_d(s(x)) = o^{(i)} \text{ iff } \pi(o^{(i)}) = x. \tag{8}$$

The episode trajectory data of driver x is $P^{(x)} = \{s_{t_0}, o_0, r_o, \cdots, s_{t_k}, o_k, r_k, s_{t_{k+1}}, \cdots, s_{t_K}\}$. We use the notation o_k for the k-th option at time t_k to differentiate it from the indexing for orders $o^{(i)}$. The option o_k may be to idle in addition to taking a trip.

Similarly for an MDP, the state-value function $V^{\pi_d}(s)$ of the semi-MDP is defined as the long-term discounted cumulative reward received throughout the options given s, following π_d:

$$V^{\pi_d}(s) := E\left[\sum_{i=1}^{K-k} \gamma^{(t_{k+i}-t_k-\tau_{o_k}-\tau_{e_k})} \hat{R}_{k+i} \middle| s_{t_k} = s\right], \tag{9}$$

where $\hat{R}$ is the random variable whose realized value is $\hat{r}$. The time component for the discount factor in front of $\hat{R}_{k+i}$ is because the reward $\hat{R}_{k+i}$ starts to be collected after the k-th transition is completed. The Bellman equation for π_d is

$$V^{\pi_d}(s) = E_{O_{disp}(s)\sim\mathcal{O}}\left[\hat{R}(s, \pi_d(s; O_{disp}(s))) + \gamma^{(\tau_e+\tau_o)} V^{\pi_d}(s')\right], \tag{10}$$

where $O_{disp}(s)$ is the set of open trip orders for dispatching (i.e., the action set) at s, and $\mathcal{O}$ is the corresponding demand distribution. We made the dependency of π_d on O_{disp} explicit by writing $\pi_d(s; O_{disp}(s))$. Following Boutilier et al. (2018), we can define the corresponding embedded semi-MDP by augmenting the state s with the realized action set $O_{disp}(s)$, $\tilde{s} := (s, O_{disp}(s)) \in \tilde{S}$ and denoting the corresponding policy $\tilde{\pi_d}$. Then we can recover the standard Bellman equation for $\tilde{\pi_d}$,

$$V^{\tilde{\pi_d}}(\tilde{s}) = E_{\tilde{s}'\sim\tilde{\mathcal{S}}}\left[\hat{R}(\tilde{s}, \tilde{\pi_d}(\tilde{s})) + \gamma^{(\tau_e+\tau_o)} V^{\tilde{\pi_d}}(\tilde{s}')\right], \tag{11}$$

where $\tilde{\mathcal{S}}$ is the conditional distribution of the state $\tilde{s}'$ given the action $\tilde{\pi_d}(\tilde{s})$. In practice, we use TD-learning to learn $V^{\pi_d}(s)$ from a collection of realized trajectories.

Similarly, the state-option value function $Q^{\pi_d}(s, o)$ is defined as the long-term discounted cumulative reward received throughout the options by following π_d given current state s and executing option o on s:

$$Q^{\pi_d}(s, o) := E\left[\sum_{i=1}^{K-k} \gamma^{(t_{k+i}-t_k-\tau_{o_k}-\tau_{e_k})} \hat{R}_{k+i} \middle| s_{t_k} = s, o_k = o\right]. \tag{12}$$

Tabular Temporal-Difference Learning

At the policy evaluation stage, V^{π_d} is learned through tabular temporal-difference (TD) learning (Sutton 1988) (TD(0) to be exact) using $\{P^{(x)}\}_{x=x^{(0)}}^{x^{(J)}}$, the trip and idle movement data for all drivers (from driver $x^{(0)}$ to $x^{(J)}$) collected for the training period, for example, a month. Algorithm 1 lays out the key steps of TD(0) applied to the semi-MDP in the previous section. The update term $r + \gamma V^{\pi_d}(s') - V^{\pi_d}(s)$ is the *TD-error* for the transition experience (s, o, r, s'), and α is the step size.

Algorithm 1 TD(0) for Driver Semi-MDP

```
Require: $\{P^{(x)}\}_{x=x^{(0)}}^{x^{(J)}}$ for the training period collected by $\pi$, $\alpha \in (0,1]$
Ensure: $V^{\pi_d}(s) = 0$, $\forall s$ whose time-bucket contains $T$.
  Initialization: $V^{\pi_d}(s) = 0$, $\forall s \in S$
  for each episode $P$ do
    for each transition $(s, o, r, s')$ do
      $V^{\pi_d}(s) \leftarrow V^{\pi_d}(s) + \alpha\left[\hat{r} + \gamma^{(\tau_o+\tau_e)} V^{\pi_d}(s') - V^{\pi_d}(s)\right]$
    end for
  end for
```

The improved system dispatching policy π' with respect to V^{π_d} is generated by the operator $\Pi(w)$ during the matching stage through the combinatorial optimization discussed earlier. The edge weights are computed as the sample (predicted) advantage (Baird 1993) of each possible match between $o^{(i)}$ and $x^{(j)}$, using V^{π_d}:

$$w_{o^{(i)},x^{(j)}}(V^{\pi_d}) := \hat{p}^{(i)} + \gamma^{(\hat{\tau}_o^{(i)}+\hat{\tau}_e^{(i)})} V^{\pi_d}(g(l_d{}^{(i)}, \hat{t}_d^{(i)})) - V^{\pi_d}(s(x^{(j)})), \tag{13}$$

with the understanding that $\hat{p}^{(i)}$ is discounted as in Equation (7). The advantage, Equation (13), can be viewed as the relative change in the long-term value with respect to the current spatiotemporal point of the driver $x^{(j)}$, should order $o^{(i)}$ be assigned to $x^{(j)}$. The generated policy $\pi' = \Pi(w(V^{\pi_d}))$ is collective-greedy with respect to V^{π_d} (through the advantage). The sample advantage admits the same form as the TD-error. We observe that the edge weight, Equation (13), penalizes long pickup distance in that both the immediate reward $\hat{p}^{(i)}$ (see Equation (7)) and the discount factor for the value term of the destination would decrease with increasing estimated en route time $\hat{\tau}_e^{(i)}$, thus lowering the advantage. The new policy is guided by the independent long-term option advantages of the drivers to approximately maximize their total income while discouraging long pickup wait times for the passengers. We formalize our RL framework for dispatching in Algorithm 2.

Remark 2. The goal of our framework is to maximize the total cumulative reward. Hence, it is natural to make the policy generated by Π collective-greedy with respect to the state-option value Q^{π_d}, that is, using $w_{o^{(i)},x^{(j)}} = Q^{\pi_d}(s(x^{(j)}), o^{(i)})$ in the combinatorial optimization, Equation (3), in the matching step. However, including options in a tabular value function would make the size of the table very large (at

Algorithm 2 Generalized Policy Iteration for Order Dispatching

```
Require: Dispatching policy $\pi$ (and corresponding $\pi_d$). Storage buffer $B$ with episode trajectory
  data $\{P^{(x)}\}$ collected by $\pi$.
  for $t = 1, 2, \cdots$ do
    Learn the value function $V^{\pi_d}$ from the data in $B$ using a policy evaluation method, e.g.,
    CVNet, TD(0).
    Compute $w(V^{\pi_d})$ by (13) and generate $\pi' = \Pi(w)$.
    Match orders and drivers with batch windows using $\pi'$.
    Collect new trip and driver trajectory data. Fill $B$ with new data.
    $\pi \leftarrow \pi'$
  end for
```

least the square of the number of spatiotemporal cells). It is possible to use the sample approximation $\hat{p}^{(i)} + \gamma^{(\hat{\tau}_o^{(i)}+\hat{\tau}_e^{(i)})} V^{\pi_d}(g({l_d}^{(i)}, {t_d}^{(i)}))$ instead, but the resulting solution would disregard the drivers' current states in this case. As we discussed in Remark 1, this would result in the loss of a desirable property that promotes fairness and helps improve overall driver experience on the platform.

Deep Reinforcement Learning

An input spatial point l to the CMAC activates a set of grids of multiple resolutions by the quantization functions $\{q_k(l)\}_k$, which generate a sparse activation vector $c(l) \in \mathbb{R}^A$ (A is the size of *conceptual memory*) that maps l to appropriate rows of the embedding matrix $M \in \mathbb{R}^{A\times m}$ by $c(l)^T M$. The embeddings are updated as part of the neural network in the learning (training) stage to learn the best feature representation of each grid. The embedding layer in conjunction with other state features (e.g., supply-demand context) is connected to a multilayer perceptron (MLP) to output the final state value. CVNet is trained within a DQN-like framework with minibatch stochastic gradient decent.

Sensitivity of the value function to input perturbation would propagate to the policy derived from the value function. To improve the robustness of CVNet to input perturbation (to which a tabular value function is susceptible when training data are sparse for certain parts of the spatiotemporal space, creating 'spikes' in the value table), we regularize an upper bound of the Lipschitz constant of V^{π_d}, $\mathcal{L}(V^{\pi_d})$, that is, to bound the output with respect to the norm of all input states. Since $V^{\pi_d}(s) = (v_L \circ v_{L-1} \circ \cdots v_1)(s)$, where $\{v_h\}_{h=1}^L$

are a series of constituent functions for the L layers of the neural network, $\mathcal{L}(V^{\pi_d}) \leq \Pi_h \, \mathcal{L}(v_h)$, and we have derived the analytical forms of the Lipschitz constants for the Cerebellar Embedding layer and the MLP layers in Tang et al. (2019).

Discussion

The system-centric value function $V^{\pi}(X, O_{disp})$ has a global information state and supports a system policy. Let $\mathbf{s} := (X, O_{disp})$, and $\mathbf{a}$ be the assignment actions for all the drivers and open orders. The Bellman optimality condition for the system-centric (global) MDP is (assuming unit-time trips)

$$V^{\pi^*}(\mathbf{s}) = \max_{\mathbf{a} \in A(\mathbf{s})} E_{\mathbf{s}'} \left[R(\mathbf{s}, \mathbf{a}) + \gamma V^{\pi^*}(\mathbf{s}') \right]. \tag{14}$$

From Equation (6), the global-view driver-centric values (Holler et al. 2019), which we denote by $V_j^{\pi}(s^{(j)}; X, O_{disp})$ for each driver, sum up to the system value with respect to the same policy, that is, $\sum_j V_j^{\pi}(s^{(j)}; X, O_{disp}) = V^{\pi}(X, O_{disp})$. In our approach, we are not solving Equation (14). Instead, we have focused on the driver view of the system dispatching policy and the driver-centric MDP. We define $\pi^{(j)}$ to be the j-th driver's view of π, such that $\pi^{(j)}(s^{(j)}, O_{disp}) = o \in O_{disp}$. We note that the partial-view driver-centric value function $V^{\pi^{(j)}}(s^{(j)}, O_{disp})$ is in the same form as the embedded MDP value function, $V^{\tilde{\pi_d}}(\tilde{s})$ in Equation (11). Under the partial view of the system policy, $\{V^{\pi^{(j)}}(s^{(j)})\}_j$ sum up to the system-centric value with respect to the same underlying system policy, given that the set of states $\{s^{(j)}\}$ have the same time component. Since $V^{\pi_d}(s)$, which is the compressed version of $V^{\tilde{\pi_d}}(\tilde{s})$ (Boutilier et al. 2018), is learned by crowdsourcing all drivers' experience trajectories, it can be thought of as the 'mean' driver-centric value function (shared by all drivers) with the corresponding 'mean' driver-view policy π_d. In general, $\sum_j V^{\pi_d}(s^{(j)})$ is only an approximation of the system value function up to an expectation over O_{disp}. Hence, the policy evaluation step is approximate in the sense that the target is not the true system value function. The policy is greedy with respect to a value function defined as the sum of the partial-view driver-centric values evaluated at each driver's state.

References

Albus JS (1971) A theory of cerebellar function. *Mathematical Biosciences* 10(1-2):25–61.

Alonso-Mora J, Samaranayake S, Wallar A, Frazzoli E, Rus D (2017) On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114(3):462–467.

Bailey WA Jr, Clark TD Jr (1987) A simulation analysis of demand and fleet size effects on taxicab service rates. *Proceedings of the 19th conference on Winter simulation*, 838–844 (Association for Computing Machinery, New York).

Baird LC III (1993) Advantage updating. Technical report, Wright Lab, Wright-Patterson Air Force Base, Dayton, OH.

Bello I, Pham H, Le QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. *ICLR 2017 Workshop Track*, URL `https://openreview.net/pdf?id=Bk9mxlSFx`, accessed May 19, 2020.

Boutilier C, Cohen A, Hassidim A, Mansour Y, Meshi O, Mladenov M, Schuurmans D (2018) Planning and learning with stochastic action sets. *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 4674–4682, URL `https://www.ijcai.org/Proceedings/2018/0650.pdf`, accessed April 21, 2020.

Brodsky I (2018) H3: Uber's hexagonal hierarchical spatial index. URL `https://eng.uber.com/h3/`, accessed on June 26, 2019.

Hales TC (2001) The honeycomb conjecture. *Discrete & Computational Geometry* 25(1):1–22.

Holler J, Vuorio R, Qin Z, Tang X, Jiao Y, Jin T, Singh S, Wang C, Ye J (2019) Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. Wang J, Shim K, Wu X, eds., *2019 IEEE International Conference on Data Mining (ICDM)*, 1090–1095 (Institute of Electrical and Electronics Engineers, Washington, DC).

Jindal I, Qin ZT, Chen X, Nokleby M, Ye J (2018) Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining. *2018 IEEE International Conference on Big Data (Big Data)*, 1417–1426 (Institute of Electrical and Electronics Engineers, Washington, DC).

Kuhn HW (1955) The hungarian method for the assignment problem. *Naval research logistics quarterly* 2(1-2):83–97.

Kümmel M, Busch F, Wang DZ (2016) Taxi dispatching and stable marriage. *Procedia Computer Science* 83 (December):163–170.

Li M, Qin Z, Jiao Y, Yang Y, Wang J, Wang C, Wu G, Ye J (2019) Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. *The World Wide Web Conference*, 983–994 (International World Wide Web Conferences Steering Committee, Geneva).

Lopes PA, Yadav SS, Ilic A, Patra SK (2019) Fast block distributed cuda implementation of the hungarian algorithm. *Journal of Parallel and Distributed Computing* 130 (August):50–62.

Miao F, Han S, Lin S, Stankovic JA, Zhang D, Munir S, Huang H, He T, Pappas GJ (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):463–478.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

National Bureau of Statistics of China (2019) National economic performance maintained within an appropriate range in 2018 with main development goals achieved. *National Bureau of Statistics of China* URL `http://www.stats.gov.cn/english/PressRelease/201901/t20190121_1645832.html`, accessed May 16, 2020.

Nazari M, Oroojlooy A, Snyder L, Takác M (2018) Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, 9839–9849.

Oda T, Joe-Wong C (2018) Movi: A model-free approach to dynamic fleet management. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2708–2716 (Institute of Electrical and Electronics Engineers, Washington, DC).

Özkan E, Ward AR (2020) Dynamic matching for real-time ride sharing. *Stochastic Systems* 10(1):29–70.

Sahr K (2011) Hexagonal discrete global grid systems for geospatial computing. *Archives of Photogrammetry, Cartography and Remote Sensing, Vol. 22, 2011, p. 363-376* 22 (January):363–376.

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. URL `https://arxiv.org/pdf/1707.06347v2.pdf`, accessed April 21, 2020.

Shou Z, Di X, Ye J, Zhu H, Zhang H, Hampshire R (2020) Optimal passenger-seeking policies on e-hailing platforms using markov decision process and imitation learning. *Transportation Research Part C: Emerging Technologies* 111 (February):91–113.

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. (2016) Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484.

Sutton RS (1988) Learning to predict by the methods of temporal differences. *Machine learning* 3(1):9–44.

Sutton RS, Barto AG (2018) *Reinforcement learning: An introduction* (MIT press, Cambridge, MA).

Sutton RS, Precup D, Singh S (1999) Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.

Tang X, Qin Z, Zhang F, Wang Z, Xu Z, Ma Y, Zhu H, Ye J (2019) A deep value-network based approach for multi-driver order dispatching. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 1780–1790 (Association for Computing Machinery, New York).

Tsitsiklis JN, Van Roy B (1996) Feature-based methods for large scale dynamic programming. *Machine Learning* 22(1-3):59–94.

Uher V, Gajdoš P, Snášel V, Lai YC, Radeckỳ M (2019) Hierarchical hexagonal clustering and indexing. *Symmetry* 11(6):731.

University of Michigan Center for Sustainable Studies (2019) Us cities factsheet. *Center for Sustainable System, University of Michigan* URL `http://css.umich.edu/sites/default/files/US\%20Cities_CSS09-06_e2019.pdf`, accessed April 21, 2020.

Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. *Thirtieth AAAI conference on artificial intelligence*, 2094–2100 (Association for the Advancement of Artificial Intelligence, Menlo Park, CA).

Verma T, Varakantham P, Kraus S, Lau HC (2017) Augmenting decisions of taxi drivers through reinforcement learning for improving revenues. *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 409–417 (Association for the Advancement of Artificial Intelligence, Menlo Park, CA).

Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. *Advances in Neural Information Processing Systems*, 2692–2700.

Wang Z, Qin Z, Tang X, Ye J, Zhu H (2018) Deep reinforcement learning with knowledge transfer for online rides order dispatching. *2018 IEEE International Conference on Data Mining (ICDM)*, 617–626 (Institute of Electrical and Electronics Engineers, Washington, DC).

Xu Z, Li Z, Guan Q, Zhang D, Li Q, Nan J, Liu C, Bian W, Ye J (2018) Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 905–913 (Association for Computing Machinery, New York).

Yan C, Zhu H, Korolko N, Woodard D (2019) Dynamic pricing and matching in ride-hailing platforms. *Naval Research Logistics* .

Zhang L, Hu T, Min Y, Wu G, Zhang J, Feng P, Gong P, Ye J (2017) A taxi order dispatch model based on combinatorial optimization. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2151–2159 (Association for Computing Machinery, New York).