9-2023

When routing meets recommendation: Solving dynamic order recommendations problem in peer-to-peer logistics platforms

Zhiqin ZHANG

Waldy JOE

Yuyang ER

Hoong Chuin LAU
Singapore Management University, hclau@smu.edu.sg

When Routing Meets Recommendation: Solving Dynamic Order Recommendations Problem in Peer-to-Peer Logistics Platforms

Zhiqin Zhang[1], Waldy Joe[1], Yuyang Er[2], and Hoong Chuin Lau[1(✉)]

[1] School of Computing and Information Systems, Singapore Management University, Singapore, Singapore
{zqzhang.2020,waldy.joe.2018}@phdcs.smu.edu.sg, hclau@smu.edu.sg
[2] AI Singapore, Singapore, Singapore
yuyang@aisingapore.org

Abstract. Peer-to-Peer (P2P) logistics platforms, unlike traditional last-mile logistics providers, do not have dedicated delivery resources (both vehicles and drivers). Thus, the efficiency of such operating model lies in the successful matching of demand and supply, i.e., how to match the delivery tasks with suitable drivers that will result in successful assignment and completion of the tasks. We consider a Same-Day Delivery Problem (SDDP) involving a P2P logistics platform where new orders arrive dynamically and the platform operator needs to generate a list of recommended orders to the crowdsourced drivers. We formulate this problem as a Dynamic Order Recommendations Problem (DORP). This problem is essentially a combination of a user recommendation problem and a Dynamic Pickup and Delivery Problem (DPDP) where the order recommendations need to take into account both the drivers' preference and platform's profitability which is traditionally measured by how good the delivery routes are. To solve this problem, we propose an adaptive recommendation heuristic that incorporates Reinforcement Learning (RL) to learn the parameter selection policy within the heuristic and eXtreme Deep Factorization Machine (xDeepFM) to predict the order-driver interactions. Using real-world datasets, we conduct a series of ablation studies to ascertain the effectiveness of our adaptive approach and evaluate our approach against three baselines - a heuristic based on routing cost, a dispatching algorithm solely based on the recommendation model and one based on a non-adaptive version of our proposed recommendation heuristic - and show experimentally that our approach outperforms all of them.

Keywords: Crowdsourced delivery · Data-driven optimization · Recommendations system

J. R. Daduna et al. (Eds.): ICCL 2023, LNCS 14239, pp. 18–35, 2023.
https://doi.org/10.1007/978-3-031-43612-3_2

1 Introduction

Peer-to-Peer (P2P) transportation platforms, such as Uber, Lyft, and FoodPanda, have been gaining popularity in recent years with greater adoption of internet technology (such as e-commerce and mobile apps) and the growing demand for more efficient urban mobility. These platforms include ride-hailing or ride-sharing and crowdsourced delivery service providers where the former provide transportation services for people while the latter for goods. The key feature of such platforms is that the platform owners do not own the physical assets (fleet of vehicles) nor employ the transport operators (drivers or riders, used interchangeably) unlike the traditional taxi companies or logistics service providers. Thus, the efficiency of such P2P transportation model lies in the successful matching of demand and supply, i.e., how to match the delivery tasks (or orders or jobs, used interchangeably) with the appropriate drivers that will result in successful assignment and completion of the jobs. This is an interesting research challenge as unlike in the traditional routing problems, the drivers have a choice to accept or reject the jobs.

The work presented in this paper is motivated by a real-world problem scenario involving a P2P logistics platform uParcel[1], where new delivery tasks are being recommended throughout the day in the form of a ranked list (also known as menu) to individual crowdsourced drivers who have already existing prescheduled delivery tasks. This problem is essentially a combination of a user recommendation problem [23] and a Same-Day Delivery Problem (SDDP) which is a variant of a Dynamic Vehicle Routing Problem (DVRP) [5]. For a successful matching of delivery tasks to drivers, the recommendations need to take into account not only the platform's profitability (which in classical SDDP is determined by how good the delivery routes are), but also the drivers' preferences for the tasks recommended.

Both, the user recommendation problem and the SDDP have been widely studied and researched in their respective fields. However, there is a great potential for synergy between the two research domains in addressing the problem presented in this paper. The traditional Recommender Systems (RS) are only able to capture users' general personal preferences [29] while the Sequential Recommender Systems (SRS) go one step further by taking into account the sequential dependencies of user-item interactions for more accurate recommendation [7]. Although an SRS takes into account the existing state of a user, it is only able to recommend the user's next action, i.e., in the context of e-commerce, the item that he or she will purchase next. This is insufficient in addressing the problem in this paper which is akin to recommending a new product given that users have purchased a list of items (pre-scheduled orders) and that they are planning to purchase another list of items in the future (new orders that arrive dynamically). Meanwhile, current works in solving the SDDP assume that the drivers

[1] uParcel is a Singapore start-up company which offers on-demand delivery and courier services for business and consumers. See https://www.uparcel.sg/ (last access date 02 July 2023).

will *always* accept the assigned delivery tasks [27,28], which is not the case in a crowdsourced setting.

To address this gap, we bring together two key ideas from Reinforcement Learning and Recommender Systems. More precisely, we propose an adaptive recommendation heuristic (ARH) that incorporates reinforcement learning (RL) to learn the parameter selection policy within the heuristic and a RS model called xDeepFM [15] to predict the driver's probability of accepting an order based on their preference. ARH relies on a linear scalarization function that takes into account both routing information and order-driver interaction in generating the recommendation, and the RL-trained policy is utilized to adaptively determine the exact weightages of this linear function at a given pre-decision state.

2 Related Works

Same-Day Delivery Problem. There are two broad categories of approaches for solving SDDP or DVRP in general namely offline or pre-processsed decision support and online decision [24]. In offline approaches, policies or values for decision-making are computed prior to the execution of a plan. Here, the problem is usually formulated and solved as an Markov Decision Process (MDP). Unfortunately, MDP-based approaches (specifically tabular-based ones) fall into the curse of dimensionality and hence are not suitable for most real-world problems [19]. Approximate Dynamic Programming (ADP) approaches [20] are commonly used to tackle the scalability issue, and one such ADP method for the DVRP is Approximate Value Iteration (AVI) [1]. Increasingly, there also have been many recent works that addressed to solve the DVRP using RL [6,8,13,14]. Meanwhile, lookahead approaches have been applied successfully to solve dynamic and stochastic routing problems. These are commonly termed as rollout algorithms, e.g., [25], the pilot method, e.g., [17] and Multiple Scenario Approach (MSA), e.g., [4,28]. The approaches proposed in these works are not directly applicable to our problem since we are dealing with crowdsourced drivers; however, like many current works that use RL to solve the DVRP, we utilize RL to learn a policy to guide our recommendation heuristic.

Recommender System. There are two broad approaches to recommender systems, namely collaborative filtering methods and content-based methods. Collaborative filtering methods rely on past-recorded interactions between users and predict through similar user patterns to produce new recommendations. One such example is Factorization Machines (FM) [22]. Meanwhile, content-based methods use item features to recommend other items similar to what the user likes and based on their previous action or explicit feedback. One such example is linear models with Follow-the-Regularized-Leader (FTRL) [16]. Lately, Deep Neural Networks have been utilised to learn high-order interactions [15]. To combine the strength of a wide linear model (i.e., Generalized Linear Models) and a deep neural network for recommender systems, Google proposed a framework called *Wide & Deep* [9]. Subsequently a wide and deep architecture of DeepFM

that integrates the Factorization Machine (the wide component) and the Multi-Layer Perceptron (the deep component) was proposed in [11]. xDeepFM, an extension of the DeepFM that can jointly model explicit and implicit feature interactions was proposed in [15]. In this paper, we select xDeepFM to predict drivers' probability of selecting an order as it combines both content-based and content filtering (hybrid approach) and mitigates the cold start and data sparsity issues by relying on the factorization of the sparse user-item matrix.

Dynamic Matching Problem. [3] introduced the concept of dynamic order matching in a Peer-to-Peer (P2P) logistics platform, which shares similarities with the problem addressed in this paper. The authors framed the problem as a two-stage decision problem and proposed a multiple scenario approach that involved sampling various driver selection scenarios and solving an integer program for each scenario to generate the final menu. However, it should be noted that the authors of [3] considered a simplified version of the DORP where drivers are only allowed to select one order, and no time windows or capacity constraints are imposed. Furthermore, the authors modeled the drivers' preferences using a pre-determined utility function. In contrast, the problem addressed in this paper takes into account real operational constraints and incorporates drivers' preferences based on historical data, making it a more realistic and comprehensive representation of the DORP.

3 Problem Description and Model

3.1 Problem Description

We assume that dynamic orders are generated throughout the planning horizon, and at a predetermined frequency, the platform consolidates the newly arrived dynamic orders and any unassigned orders and generates a personalized order recommendation list for the crowdsourced drivers. The dynamic orders consist of pickup and delivery tasks, order size, time window requirements for both pickup and delivery, and an expiration time. If a driver arrives at the pickup or delivery location earlier than the specified time window, a waiting time will be incurred. In the event that no driver accepts an order and arrives at the pickup location after the specified time limit, the order will be automatically canceled. A simple illustration of the proposed problem can be found in the supplementary materials[2].

The objective of the problem is to maximize the number of orders fulfilled within the given planning horizon, specifically, a full working day. This objective can be achieved by ensuring that the order recommendation list considers both the driver's preference and routing considerations. The driver's preference ensures that they have a high probability of selecting at least one of the recommended orders, while the routing considerations ensure that drivers with optimal delivery routes can fulfill more orders while minimizing delivery costs.

[2] https://anonymous.4open.science/r/iccl2023-7ADC/SupplementaryMaterials.pdf.

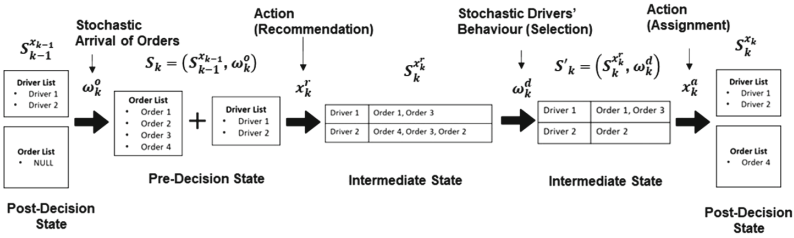

Fig. 1. Sequential decision process of a DORP.

3.2 Model Formulation

This problem can be modeled as a sequential decision process with stochastic information [20]. A visual depiction of the entire decision process is presented in Fig. 1. Here, we present an MDP formulation of the problem.

Decision Epoch. The decision epoch in our model is time-based. We discretize the planning horizon (e.g., a day) into multiple time periods. Here, we use $k = 1, 2 \ldots, i, i+1 \ldots, n$ to represent each decision point.

State. As shown in Fig. 1, a state consists of two parts, pre-decision state S_k and post-decision state $S_k^{x_k}$. Both states are represented by the same tuple $\langle d_k, o_k \rangle$ where d_k denotes the list of drivers and o_k denotes the list of unassigned orders. The list of drivers can be further broken down as follows: $d_k = \left\{d_k^1, \ldots, d_k^{m_k}\right\}$, where m_k denotes the number of the active drivers at decision epoch k. We assume the drivers are active when they arrive at pickup or delivery nodes within this decision time slot. Each element in d_k consists of the following features: the last reported location, vehicle capacity and the route of the remaining orders. Similarly, the list of unassigned orders is denoted as $o_k = \left\{o_k^1, \ldots, o_k^{n_k}\right\}$, where n_k denotes the number of the unsigned orders at decision epoch k. Each element in o_k consists of the following features: pickup and delivery locations, time windows for pickup and delivery, and size of the order.

Action/Decision. The decision in this model refers to the action of generating personalized order recommendation lists for each driver. Although in reality, the platform needs to execute an assignment action once drivers make their selections (see Fig. 1), we assume that this assignment is governed by a predetermined rule and we treat this step as part of the environment. We denote our decision variable as $x_k = \left\{x_k^1, \ldots, x_k^{m_k}\right\}$, where m_k still denotes the number of the riders available in this time slot. Each element x_k^i in x_k contains a subset of unassigned orders a sorted in descending order of suitability for driver i. Note that each order may be recommended to more than one driver and each driver can have more than one recommended order.

Transition. There are two main transitions in this model. Firstly, the transition occurs from the previous post-decision state, denoted as $S_{k-1}^{x_{k-1}}$, to the pre-decision state, represented as S_k. This transition is triggered by the realization of new dynamic orders, denoted as ω_k. The second transition takes place from

the pre-decision state to the post-decision state, which occurs after executing the action x_k. However, this second transition introduces additional complexity due to multiple uncertainties that influence the outcome of the action taken. Specifically, there are two intermediate states that arise from two sources of uncertainty, namely the selection behaviors of the drivers and the assignment rule.

Reward/Objective Function. The reward $R(S_k, x_k)$ of an action x_k is defined as the total number of orders fulfilled for the decision epoch k. The solution to this Markov Decision Process (MDP) is a policy that generates the menus at each state. The optimal solution is the policy that maximizes the current reward and the total expected rewards for future states.

4 Solution Approach

Generating the personalized order recommendation lists directly based on the model formulation presented in the previous section will involve an enumeration of all possible permutations of unassigned orders of variable length for each driver. Even if we limit the length of the lists, the action space will still be combinatorial. Thus, to address the curse of dimensionality, we design a heuristic approach to generate the recommendation lists.

The remaining part of this section is organised as follows. We first introduce how our proposed ARH generates the recommendation lists. We then explain the two key components within the ARH, namely how RL is used to train the weightage selection policy for this heuristic; and how the drivers' preference is modelled and learnt from historical data by using xDeepFM.

4.1 Adaptive Recommendation Heuristic

In DORP, the state space and action space are too large to enumerate for large-scale problems. Compared with most dynamic same-day delivery problems such as [10,12] where a decision involves the assignment of one dynamic order to one rider, our problem involves a multi-order multi-driver matching, which leads to a very large state space. In addition, the action space also faces the "curse of dimensionality". Unlike the action space in [3], the recommendation lists for riders in DORP need to be sorted by the relevance (rider-order relevance) which contributes to a larger action space. This task is even made more challenging as decisions need to be done in real-time. Thus, we propose a heuristic solution approach to aggregate the action space.

Heuristic Scoring Function. In our heuristic, the key idea is to compute a driver-order pair score based on the routing cost and riders' personal preference. For a given rider-order pair score, the higher this score is, the more suitable for this rider to serve this order by our consideration. The recommendation list for each rider is then generated by these scores. The function to calculate the driver-order score is given as follows:

$$score(d_i, o_j) = \alpha H_1 + (1 - \alpha) H_2 \tag{1}$$

where α and $1 - \alpha$ are the weights on H_1 and H_2, respectively. The value H_1 is the estimated cost for inserting this order to this rider's route, calculated by the distance of the pickup location of this order to the nearest location of the remaining orders carrying by this rider (referred to as D_n) and normalized between 0 and 1:

$$H_1 = \frac{D_{max} - D_n}{D_{max}} \tag{2}$$

where D_{max} is the largest distance value for any two locations in a given problem instance.

The value of H_2 is the probability of the prediction for this rider's preference for this order, and computed using our proposed xDeepFM model where the details will be presented in Sect. 4.3.

Balancing Platform Objective and Driver Preferences. As a recommender system, the challenge is in calibrating the values of α adaptively over decision epochs, as the environment changes, in order to ensure good profitability (jobs served) on one hand, and good acceptance rate by drivers on the recommended jobs on the other.

From the point of view of the logistics platform, the efficiency of the dynamic routing plan hinges on timeliness of deliveries which in turn is measured by the physical measures such as the routing cost, as proposed in the SDDP and DVRP literature. On the other hand, from the drivers' point of view, they would like to serve orders of their preferences. For instance, the drivers may be more willing to serve the orders in the region which they are more familiar with, even though the locations may be a little far away from their current location and/or the locations of their pre-scheduled orders. If the heuristic generates the menu based on the preferences of the riders, it will not only improve the acceptance of these orders but also the user experience for drivers.

To achieve a good balance of these two potentially conflicting measures, we propose our ARH. The key idea is that we dynamically adjust the weights for each selection criterion based on the current state when making the decision. Intuitively, if the delivery resources are insufficient in some time period, the heuristic should assign orders to focus on the operational efficiency. Otherwise, we want to recommend preferred options to riders, hoping that the orders recommended are more likely accepted by riders (according to their personal preference). In the following, we present a Reinforcement Learning approach to obtain the parameter values (weights) for balancing these two selection criteria.

Finally, we present our heuristic approach, ARH, to calculate driver-order scores that take into account both driver preferences and routing costs at every decision epoch.

ARH consists of the following components:

– Driver's predicted preference score to represent the drivers' selection behaviour based on historical data.

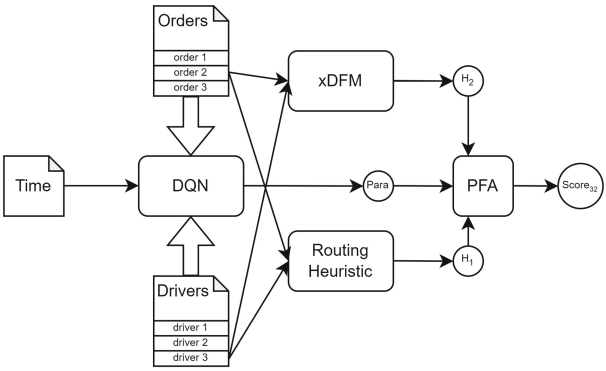

Fig. 2. Overview of the proposed Adaptive Recommendation Heuristic approach.

- A routing heuristic to calculate the cost for a given driver to serve a given order.
- RL-trained policy to determine the weights attached to each objective at a given pre-decision state.

Our proposed heuristic is based on a Policy Function Approximation (PFA) approach [21]. Traditional PFAs are analytical functions by which information in the state variable is mapped directly to a decision. The information of the state in this problem includes two perspectives: platform and drivers. This PFA applies Eq. 1 to calculate the pair scores to generate the final decision, namely the recommendation lists.

The overview of ARH can be found in Fig. 2. The two hollow arrows represent the input of all orders and drivers' data into the Deep Q-Network (DQN), while the other solid arrows signify the transfer of a single value. More details about the algorithm showing how the ARH generates the recommendation lists in a given decision epoch can be found in supplementary materials.

4.2 RL for Selection in ARH

In order to select a proper heuristic for a given decision epoch, we propose an RL approach. Before we elaborate this approach, we describe the MDP formulation for the new decision problem after introducing our proposed heuristic.

MDP Reformulation with State and Action Space Aggregation. The state of this new MDP is an aggregated state. Then we use handcrafted features to calculate the rider-order score. To condense the new state to a set of features, we select the following state information for our new MDP:

- Current time t for this decision epoch k. As the authors of [6,26] select the current point of time of this decision epoch, we also include this feature in our representation of state.

- Total number of unassigned orders n_k and active drivers m_k in this time slot (or decision epoch). We select these two features because it can help to capture how dense the supply/demand in the current time slot is. Combined with the current time, these two features can provide the information to learn the policy for making decisions not only on the current decision epoch but also looking further in terms of supply/demand ratio.
- Information about riders. We do not need all the rider's information in the original MDP. Here we consider to use the remaining orders (or carrying orders) of riders. Based on the remaining orders, we can calculate the distance and time-related information of a given new order to those remaining orders. These are main features to capture the power of supply in the near future time slots. For example, for driver i, the committed delivery time of carrying orders is denoted as vector $\boldsymbol{to_i}$. Then the average bottleneck time for all drivers tb can be calculated by

$$tb = \frac{1}{m_k} \sum_{i=1}^{m_k} (\|\boldsymbol{to_i}\|_{-\infty} - t) \tag{3}$$

  where the negative infinite norm can give the earliest committed delivery time of all the on-going orders of a given driver.
- Information about orders. Apart from the total number of the unassigned orders to calculate the ratio of supply/demand, we also consider average remaining time to obtain the information of the demand. To be more specific, the average remaining time tr is the average of the differences between the expired time te_j for a given order j and the current time t.

$$tr = \frac{1}{n_k} \sum_{j=1}^{n_k} (te_j - t) \tag{4}$$

The action is to select one heuristic to calculate the rider-order pair score for this current decision epoch. To be more specific, we use five parameters in our proposed heuristic. These five values of α are 0, 0,25, 0.5, 0.75 and 1, respectively.

The transition and reward function remain the same as the original MDP. Then, we formulate this problem and use a RL approach to train a policy to select a proper heuristic to calculate the rider-order score in each decision epoch. By these scores, we can directly generate the personalized recommendation lists to riders.

Deep Q-Learning for Weight Selection. For a given state S_k, the action x_k is to select the most proper heuristic to generate the personalized recommendation lists for riders. Thus, the solution of the DORP is to learn a policy $\pi \in \Pi$ that selects the action for each state. The optimal solution $\pi*$ can maximize the total reward. Q-learning [30] learns a value $Q(S_k, x)$ for each state and each action pair. This Q-value can estimate the immediate reward plus the expected future rewards if the action x is taken to the state S_k. Definitely, we cannot calculate and learn all these state-action pairs due to the size of the state and action space (even after aggregation). Thus, we use the DQN approach to estimate this value.

We applied the DQN with experience replay approach in [18]. To calculate the loss function to train this Q-network, the reward function is defined as the proportion of accepted jobs in a specific decision epoch (time slot). The hyper-parameters we used can be seen in the supplementary materials.

4.3 xDeepFM for Preference Prediction

In this subsection, we introduce xDeepFM to predict the job-driver preference score which will be used by the ARH presented above.

Similar to a typical RS problem, the task is to recommend delivery jobs to drivers. The role of the xDeepFM model is to generate the value of H_2, a score for each driver-job pair that indicates the probability that driver $d_k^{m_k}$ will accept job $o_k^{n_k}$. This probability will subsequently be used by the ARH to take into account the driver's preference in selecting a job.

Architecture. The architecture of xDeepFM can be broken down into four parts: (1) embedding layer, (2) linear/FM using the raw input features, (3) Deep Neural Network (DNN) using the dense feature embeddings, and (4) Compressed Interaction Network (CIN) using the dense feature embeddings.

The output of parts (2) to (4) jointly contribute to a shared sigmoid output. The embedding layer takes in the FM inputs and represents them as lower-dimensional vectors. Linear/FM uses matrix factorization to learn the low-dimensional representations and then learns the linear regression weights for the FM layer. For higher-order interactions, these are captured by DNN and CIN where DNN learns implicit high-order feature interactions at the bit-wise level while CIN learns explicit high-order feature interactions fashion at the vector-wise manner. CIN and plain DNNs can complement each other to make the model stronger by combining these two structures. In addition, xDeepFM is customizable as it can be configured as a classical FM model by enabling the linear part and the FM part only if the dataset does not require high-order feature interactions. The architecture of xDeepFM can be seen in supplementary materials. See [15] for more details.

Feature Engineering. The features used for training the xDeepRM model are presented in supplementary materials.

In DORP we face two key challenges involved in the learning task, namely, data sparsity and highly dynamic input features. The data is sparse because the platform only records positive samples, i.e., the accepted orders but not orders that are rejected. In addition, some important input features such as the current location of the drivers and the current capacity of the vehicle are highly dynamic and play an important role in determining whether a driver will accept or reject an order. We address these challenges as follows.

Distance-Based Feature. The current location of a driver plays an important role in determining whether he or she will accept an order. Thus, inclusion of distance-based features would improve the model's performance. These features capture the distance from the driver's current location to the pickup point and

from the pickup point to the dropoff point. The challenge is the former. Unlike in typical RS, this feature is highly dynamic. The main challenge lies in calculating this value in real time as there could be some error in calculating expected versus actual values since such highly dynamic data may not be available or accurate. If it is not handled properly, these errors could accumulate over time, resulting in incorrect predictions. To overcome this, we propose to use a proxy feature, i.e., last known location to represent the current location of the driver in order to calculate the distance of the current location to the pickup location. During training, the last known location of a driver is defined as the location at the time point when jobs have last been accepted by this driver.

Negative Sampling. The given dataset consists of historical jobs accepted by individual drivers (item features), as well as a list of drivers (user features). For training the model, the accepted jobs form the positive training samples. As the platform does not allow drivers to reject jobs, all non-accepted jobs are automatically considered implicit negative samples. Hence most samples recorded are positive use cases. Thus, implicit negative samples are generated to ensure a more balanced data during model training. However, not all negative samples are useful and they may in turn introduce noise, hence negative sampling was used to decrease the amount of negative use cases. The chosen approach is by filtering within a time window and computing distance relative to the driver's most recent job/known location. We made the assumption that an active driver who accepted order(s) or performing a job at a given time window is likely to have seen and "rejected" the non-accepted jobs at the same time window.

5 Experiments

We evaluate our proposed approach using a real-world dataset that we collected from a local logistics platform. We ensure that the problem settings and input data closely resemble the real-world scenario.

5.1 Benchmark Algorithms

We present three baseline algorithms as benchmarks: one based solely on routing cost, one based solely on the driver's preference, and one that is a linear combination of these two aspects with equal weightage.

- *RH.* The first baseline algorithm is a routing heuristic (RH), which is similar to the order dispatching algorithm which uParcel currently adopts. This algorithm only considers the distance between the driver's current location and the new order location. This can be done by setting the $\alpha = 1$ in Eq. 1.
- *DP.* The second baseline algorithm is one which solely relies on the driver's preference (DP). This is similar to a typical recommender system which learns driver-order score to represent the driver's preference over a certain order. Section 4.3 describes how this preference score is computed. Then, we set the $\alpha = 0$ in Eq. 1 to ensure that the order dispatching is solely based on the drivers' preference.

– *FWH.* The third baseline algorithm is considering both aspects in a fixed weight. This is done by setting α in Eq. 1 as fixed value, (e.g. setting it as 0.5 as the third baseline algorithm). Thus, this model is a fixed weight heuristic (FWH) model where we assume a 'balanced' consideration.

5.2 Experiment Design

Data. In this experiment, we use a total of 894,794 rows of pickup-delivery tasks collected from 2021 to 2022 provided by uParcel. To train the xDeepFM model and the DQN, we generate two datasets from the raw data. More details about the dataset can be found in the supplementary materials.

xDeepFM Module. As mentioned in Sect. 4.3, we use the xDeepFM model to predict preference of the drivers. Here, we describe the detailed of our implementation of this model.

xDeepFM Training. The challenge of training the xDeepFM in this problem context lies in the sparsity of data and the dynamicity of the problem. Unlike in the typical RS, recommendation or preference scores need to be computed quickly and sometimes the input features change dynamically and may not be available at the moment of inference. This is because xDeepFM is customizable depending on the nature of the data used. We conducted an ablation study to ascertain which customized architectures of xDeepFM are effective in predicting drivers' preference. In addition, given that the current location of a driver plays an important role in determining how likely an order will be accepted, we include a distance feature to act as a proxy to determine how far the driver is from the order. Here, we use a proxy of the driver's real location because in practice, the exact location of a driver may not be available or be updated in the system due to the highly dynamic nature of this feature. See Table 1 for the list of customized models evaluated.

Model and Training Setup. The xDeepFM model that we implemented includes a DNN with 4 layers and a CIN with 3 layers. We use optuna [2] to tune the number of nodes in each layer. Apart from these hyperparameters, the training epochs is set as 500, the batch size is 128 and the learning rate is 0.001. To train the model for binary classifications, we use the following loss function:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i log\hat{y}_i + (1 - y_i)log(1 - \hat{y}_i) \tag{5}$$

where N is the number of total training instances, y_i and $\hat{y}_i$ is predicted value 0 or 1 (accept or not) and the related label.

Metrics. To evaluate the performance of the various baseline models, we use Precision@k and Recall@k where k refers to the top-k order recommendations given to a driver. To be more specific, the precision@k is defined as the number of recommended orders at top-k that are accepted by a given driver divided by k which is the number of top-k orders the model recommended to this driver.

$$Precision@k = \frac{\#\ of\ recommended\ orders\ @k\ that\ are\ accepted}{\#\ of\ recommended\ orders\ @k} \tag{6}$$

The recall@k is the proportion of accepted orders found in the top-k recommendations.

$$Recall@k = \frac{\#\ of\ recommended\ orders\ @k\ that\ are\ acceptd}{total\ \#\ of\ accepted\ orders} \tag{7}$$

We choose $k = 3$ and 5 as these are realistic lengths of a recommendation list that are both not too long (difficult to make decision) but at the same time give drivers enough options to choose.

RL Module. As mentioned in Sect. 4.2, we train a DQN to select the weights for our proposed solution method, Adaptive Recommendation Heuristic. Here, we show the details related to this module.

RL Environment. To conduct the experiments, we build a simulator for the DORP environment which simulates the selection behavior of riders. Here, we use a multi-attribute utility model similar to the one proposed in [3]. There are two attributes in this multi-attribute utility model namely detoured distance and drivers' preference, which are two considerations in decisions for our proposed DORP. In addition, we also introduce a discount factor to reflect the position of the recommended order in the list:

$$v_{i,j} = \gamma_p(\delta_1 D_{i,j} + \delta_2 P_{i,j}) \tag{8}$$

where $v_{i,j}$ is the utility value of order j to driver i. γ_p is the position discount factor. This number from large to small represents the position of the order appearing in the menu from top to bottom. δ_1 and δ_2 are two weights of the two attributes mentioned above and they sum up to one. To capture the difference of the driver selecting behavior, we uniformly randomize these two figures from driver to driver. $D_{i,j}$ is the estimated detoured distance for driver i to serve order j. Here we use the H_1 value to estimate the routing cost value. $P_{i,j}$ is the predicted probability that driver i will accept order j, which can be estimated by our trained xDFM model. A high value for $v_{i.j}$ indicates high willingness of driver i to serve the order j. To conduct the experiments, we also borrow the idea from [3] to set a threshold (0.5 in this environment to get a reasonable results by the similar current algorithm implemented in the logistics platform). Other experiment setups can be seen in the following paragraph.

Experiment Setup. The hyperparameters of the DQN can be found in supplementary materials. We train the DQN for ARH on the two-month dataset and test the performance of the ARH on the two-week dataset. More details related to the dataset can be found in supplementary materials. We define each day (or each instance) as an "episode" in the training and testing because the problem we consider is same-day delivery. We split each episode into 144 time steps (or decision epochs), where each step is 10 min.

Metrics. We evaluate our proposed ARH against the three baseline algorithms based on the test dataset and we use the order fulfill rate to compare the performance. The order fulfill rate can be calculated by:

$$order\ fulfill\ rate = 1 - \frac{\#\ canceled\ order}{\#\ total\ order} \tag{9}$$

Table 1. xDeepFM model with additional distance features outperforms the other baseline models in both precision and recall measures.

Model	Precision@3	Precision@5	Recall@3	Recall@5
FM only	0.431	0.436	0.109	0.110
FM + DNN	0.469	0.489	0.335	0.336
FM + CIN	0.468	0.478	0.095	0.095
FM + CIN + DNN	0.491	0.506	0.336	0.324
FM + CIN + DNN				
w/distance	**0.557**	**0.578**	**0.352**	**0.349**

5.3 Experiment Results and Discussion

Firstly, we show the evaluation results of the prediction model. Then, we compare the results of our ARH method with three baseline models described in Sect. 5.1.

xDeepFM. Figure 3 visualizes the convergence of the models with different components. In each graph, the horizontal axis is the epochs, while the vertical axis is the log loss. The blue and orange lines show the loss in the training data set and test data set, respectively. From these graphs, we can find that the FM model cannot predict a good result as the loss on test data set converge to about 1.25 to 1.5 after training. If introducing the DNN, the test loss of the prediction model will reduce to about 0.6 but it is not stable, comparing with the FM and the CIN model. The final prediction model, xDeepFM with all components can get the best results as the test loss converge to the smallest value.

Table 1 also shows the results that xDeepFM with the full suite of components outperforms the classical FM or FM with the subsets of the components. In addition, inclusion of distance-based feature improves the prediction.

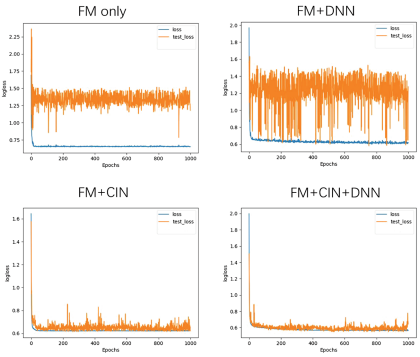

Fig. 3. Convergence of different prediction models. (Color figure online)

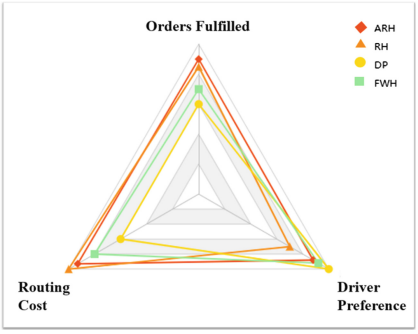

Fig. 4. Summary of the performance of different algorithms.

Table 2. Experiments results of test data set.

Instance	RH	DP	FWH	ARH
Instance 1 (912)	**93.97**%	84.87%	86.51%	93.86%
Instance 2 (383)	97.91%	98.17%	98.43%	**98.69**%
Instance 3 (779)	89.47%	83.44%	85.49%	**95.25**%
Instance 4 (467)	79.87%	72.38%	76.45%	**90.15**%
Instance 5 (453)	**100**%	**100**%	**100**%	**100**%
Instance 6 (435)	98.39%	94.48%	96.55%	**98.62**%
Instance 7 (1065)	85.45%	78.87%	80.01%	**91.27**%
Instance 8 (920)	94.02%	86.09%	87.61%	**97.83**%
Instance 9 (887)	93.46%	86.70%	87.15%	**94.59**%
Instance 10 (752)	99.20%	89.89%	90.69%	**99.34**%
Instance 11 (782	93.61%	86.57%	90.41%	**97.31**%
Instance 12 (384)	**97.14**%	**97.14**%	**97.14**%	96.89%
Instance 13 (440)	**90.68**%	87.73%	90.23%	90.23%
Instance 14 (424)	83.19%	**86.56**%	86.08%	84.67%
Avg.	92.59%	88.06	89.49%	**94.91**%
Var.	0.39%	0.57%	0.47%	**0.20**%

ARH. After having trained 200 episodes (one episode includes 144 epochs) on the training dataset, our proposed ARH with the trained DQN policy can achieve better results on most test datasets. Table 2 shows the results of the performance of our proposed ARH and three baseline algorithms. The entries are the order fulfilled rate defined as Eq. 9, so larger numbers represent better performance. The first column is instances in the two weeks data set. The number in the brackets represents the total number of the dynamic orders in one instance. Columns two to four are the results of three baseline models described in Sect. 5.1. The fifth column shows the best results among these three baselines. The last column shows the performance of our proposed ARH. From this table, we can observe that our proposed ARH can get the best results in 10 instances. ARH is the worst only in instance 12 with a slight difference. For the remaining instances, the ARH can outperform some of the baselines. As the statistical results shows, the average order fulfill rate of the ARH on the test dataset is the highest and the its variance is the smallest.

Apart from the order fulfill rate, we present a summary of the comparison on two other metrics (namely, the total routing cost and the driver preference measured by the probability of the driver accepting the order recommended to him/her based on the driver's historical data). Figure 4 shows that for routing cost, RH (i.e., solely considering routing cost) achieves the best performance compared with the other three algorithms (which is expected). Similarly, DP performs best in terms of driver preference compared with the other three

approaches. Interestingly, our approach ARH achieves the best performance in terms of number of orders fulfilled, while not compromising by a large margin in terms of the other two conflicting metrics.

All in all, the results demonstrate that our proposed framework effectively dispatches more dynamic orders for P2P platforms, thereby enhancing customer satisfaction for both riders and customers. This positive impact translates into increased profitability for the platform in both the short and long term.

Acknowledgements. This research project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (Award No: AISG2-100E-2021-089). We like to thank uParcel and AI Singapore for data, domain and comments, the ICCL PC chairs and reviewers, with special mention of Stefan Voss, for suggestions and meticulous copy-editing during the review process.

References

1. Agussurja, L., Cheng, S.F., Lau, H.C.: A state aggregation approach for stochastic multiperiod last-mile ride-sharing problems. Transp. Sci. **53**(1), 148–166 (2019). https://doi.org/10.1287/trsc.2018.0840
2. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2623–2631 (2019). https://doi.org/10.1145/3292500.3330701
3. Ausseil, R., Pazour, J.A., Ulmer, M.W.: Supplier menus for dynamic matching in peer-to-peer transportation platforms. Transp. Sci. **56**(5), 1304–1326 (2022). https://doi.org/10.1287/trsc.2022.1133
4. Bent, R.W., Van Hentenryck, P.: Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Oper. Res. **52**(6), 977–987 (2004). https://doi.org/10.1287/opre.1040.0124
5. Berbeglia, G., Cordeau, J.F., Laporte, G.: Dynamic pickup and delivery problems. Eur. J. Oper. Res. **202**(1), 8–15 (2010). https://doi.org/10.1016/j.ejor.2009.04.024
6. Chen, X., Ulmer, M.W., Thomas, B.W.: Deep Q-learning for same-day delivery with vehicles and drones. Eur. J. Oper. Res. **298**(3), 939–952 (2022). https://doi.org/10.1016/j.ejor.2021.06.021
7. Chen, X., et al.: Sequential recommendation with user memory networks. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, pp. 108–116 (2018). https://doi.org/10.1145/3159652.3159668
8. Chen, Y., et al.: Can sophisticated dispatching strategy acquired by reinforcement learning? In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1395–1403 (2019). https://doi.org/10.48550/arXiv.1903.02716
9. Cheng, H.T., et al.: Wide & deep learning for recommender systems. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 7–10 (2016). https://doi.org/10.1145/2988450.2988454
10. Dayarian, I., Savelsbergh, M.: Crowdshipping and same-day delivery: employing in-store customers to deliver online orders. Prod. Oper. Manag. **29**(9), 2153–2174 (2020). https://doi.org/10.1111/poms.13219

11. Guo, H., Tang, R., Ye, Y., Li, Z., He, X.: DeepFM: a factorization-machine based neural network for CTR prediction. arXiv preprint arXiv:1703.04247 (2017). https://doi.org/10.48550/arXiv.1703.04247
12. Hou, S., Gao, J., Wang, C.: Optimization framework for crowd-sourced delivery services with the consideration of shippers' acceptance uncertainties. IEEE Trans. Intell. Transp. Syst. **24**(1), 684–693 (2022). https://doi.org/10.1109/TITS.2022.3215512
13. Joe, W., Lau, H.C.: Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, pp. 394–402 (2020). https://doi.org/10.1609/icaps.v30i1.6685
14. Li, X., et al.: Learning to optimize industry-scale dynamic pickup and delivery problems. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 2511–2522. IEEE (2021). https://doi.org/10.1109/ICDE51399.2021.00283
15. Lian, J., Zhou, X., Zhang, F., Chen, Z., Xie, X., Sun, G.: xDeepFM: combining explicit and implicit feature interactions for recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1754–1763 (2018). https://doi.org/10.1145/3219819.3220023
16. McMahan, H.B., et al.: Ad click prediction: a view from the trenches. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1222–1230 (2013). https://doi.org/10.1145/2487575.2488200
17. Mendoza, J.E., Castanier, B., Guéret, C., Medaglia, A.L., Velasco, N.: Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands. Transp. Sci. **45**(3), 346–363 (2011). https://doi.org/10.1287/trsc.1100.0353
18. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
19. Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L.: A review of dynamic vehicle routing problems. Eur. J. Oper. Res. **225**(1), 1–11 (2013). https://doi.org/10.1016/j.ejor.2012.08.015
20. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality, vol. 842. Wiley, Hoboken (2011)
21. Powell, W.B.: Designing lookahead policies for sequential decision problems in transportation and logistics. IEEE Open J. Intell. Transp. Syst. **3**, 313–327 (2022). https://doi.org/10.1109/OJITS.2022.3148574
22. Rendle, S.: Factorization machines. In: 2010 IEEE International Conference on Data Mining, pp. 995–1000. IEEE (2010). https://doi.org/10.1109/ICDM.2010.127
23. Resnick, P., Varian, H.R.: Recommender systems. Commun. ACM **40**(3), 56–58 (1997)
24. Ritzinger, U., Puchinger, J., Hartl, R.F.: A survey on dynamic and stochastic vehicle routing problems. Int. J. Prod. Res. **54**(1), 215–231 (2016). https://doi.org/10.1080/00207543.2015.1043403
25. Secomandi, N.: A rollout policy for the vehicle routing problem with stochastic demands. Oper. Res. **49**(5), 796–802 (2001). https://doi.org/10.1287/opre.49.5.796.10608
26. Ulmer, M.W., Thomas, B.W.: Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. Eur. J. Oper. Res. **285**(1), 183–195 (2020). https://doi.org/10.1016/j.ejor.2019.04.029

27. Ulmer, M.W., Thomas, B.W., Mattfeld, D.C.: Preemptive depot returns for dynamic same-day delivery. EURO J. Transp. Logist. **8**(4), 327–361 (2019). https://doi.org/10.1007/s13676-018-0124-0
28. Voccia, S.A., Campbell, A.M., Thomas, B.W.: The same-day delivery problem for online purchases. Transp. Sci. **53**(1), 167–184 (2017). https://doi.org/10.1287/trsc.2016.0732
29. Wang, S., Hu, L., Wang, Y., Cao, L., Sheng, Q.Z., Orgun, M.: Sequential recommender systems: challenges, progress and prospects. In: 28th International Joint Conference on Artificial Intelligence, IJCAI 2019, pp. 6332–6338 (2019). https://doi.org/10.24963/ijcai.2019/883
30. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3), 279–292 (1992). https://doi.org/10.1007/BF00992698