Complex & Intelligent Systems (2021) 7:1207–1222
https://doi.org/10.1007/s40747-021-00288-y

ORIGINAL ARTICLE

Solving two-stage stochastic route-planning problem in milliseconds via end-to-end deep learning

Jie Zheng[1] · Ling Wang[1] · Shengyao Wang[2] · Yile Liang[2] · Jize Pan[2]

Received: 8 December 2020 / Accepted: 25 January 2021 / Published online: 14 February 2021

Abstract
With the rapid development of e-economy, ordering via online food delivery platforms has become prevalent in recent years. Nevertheless, the platforms are facing lots of challenges such as time-limitation and uncertainty. This paper addresses a complex stochastic online route-planning problem (SORPP) which is mathematically formulated as a two-stage stochastic programming model. To meet the immediacy requirement of online fashion, an end-to-end deep learning model is designed which is composed of an encoder and a decoder. To embed different problem-specific features, different network layers are adopted in the encoder; to extract the implicit relationship, the probability mass functions of stochastic food preparation time is processed by a convolution neural network layer; to provide global information, the location map and rider features are handled by the factorization-machine (FM) and deep FM layers, respectively; to screen out valuable information, the order features are embedded by attention layers. In the decoder, the permutation sequence is predicted by long-short term memory cells with attention and masking mechanism. To learn the policy for finding optimal permutation under complex constraints of the SORPP, the model is trained in a supervised learning way with the labels obtained by iterated greedy search algorithm. Extensive experiments are conducted based on real-world data sets. The comparative results show that the proposed model is more efficient than meta-heuristics and is able to yield higher quality solutions than heuristics. This work provides an intelligent optimization technique for complex online food delivery system.

Keywords Stochastic online route planning · Intelligent optimization · End-to-end deep learning · Supervised learning · Iterated greedy search

✉ Ling Wang
wangling@tsinghua.edu.cn

Jie Zheng
j-zheng18@mails.tsinghua.edu.cn

Shengyao Wang
wangshengyao@meituan.com

Yile Liang
liangyile@meituan.com

Jize Pan
panjize@meituan.com

1 Department of Automation, Tsinghua University, Beijing 100084, China

2 Meituan, Beijing 100102, China

Introduction

With the prevalence of mobile Internet, online food delivery (OFD) APPs have become more and more popular for the convenience in daily life. Millions and billions of transactions are completed via these APPs every day. In 2016, the worldwide market of food delivery reached up to €83 billion [39]. In China, one of the best-known OFD platforms, Meituan, obtained a total revenue of ¥24.7 billion for the second quarter of 2020. Over 457 million customers order food on Meituan platform with more than 6.3 million active restaurants to choose [23]. In the United States, the total food sales of OFD were expected to grow by 16% from 2017 to 2022 according to Morgan Stanley Research [24]. With huge market opportunities and strong user demand, the OFD will continue to develop quickly and steadily in the future.

The major mode of the Meituan is shown in Fig. 1. When a customer orders food, the order will be pushed to the corresponding restaurant and then assigned to a rider instantly

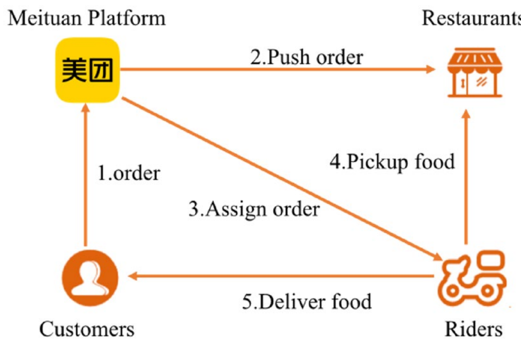

Fig. 1 The major mode of Meituan platform

with a well-planned route by the platform. The whole mode can be abstracted as an order dispatching problem and an online route-planning problem (ORPP), where the latter is the key problem of the system. The quality of the planned routes can directly influence the assignment of orders to riders where improper assignments will cause great waste of transportation resources. Besides, low-quality routes will make riders to take a detour or deliver food later than the estimated time of arrival (ETA) promised to customers, and therefore affect the efficiency of riders and experience of customers. Since there are massive orders every day and each order needs to be delivered in a short period of time (usually less than 40 min), the platform should make decisions very fast even within 1 min. The computational time left for route planning will be limited to millisecond level. In consequence, it is absolutely necessary for the platforms to put forward intelligent techniques to deal with the complex problems efficiently and robustly.

Extensive research works about solution methods [1, 29] have been carried out on the traditional route-planning problem (RPP) which is abstracted as traveling salesman problem (TSP), such as branch and bound method [3], 2-opt algorithm [34], genetic algorithm (GA) [12], and ant colony optimization [13]. Compared to the traditional RPP or TSP, the ORPP is much more complex. In addition to the immediacy requirement of the online fashion, the ORPP is also subject to time-window constraints, precedence constraints, and so on. However, few research works have been carried out on the ORPP. To minimize the total cost of the ORPP, Wang et al. [37] proposed an iterated greedy algorithm (IG) with several problem-specific heuristics. To speed up the initialization process of the same problem, they employed the extreme gradient boosting method to adaptively select the appropriate constructive algorithms [36]. The problem related to the ORPP is the single vehicle pickup and delivery problem with time windows (SVPDPTW), which is an extension of TSP and a basic version of pickup and delivery problem (PDP). Hosny and Mumford [17] presented a GA with a duplicate gene encoding to deal with a large number of constraints. To solve the SVPDPTW with capacity constraints, Edelkamp and Gath [9] designed a nested Monte Carlo search with policy adaptation. However, these algorithms cannot satisfy the immediacy requirement of online optimization. On the contrary, machine learning (ML) turns out to be promising for solving combinational optimization (CO) problems effectively in short computational time. Bengio et al. [5] have investigated the major methods of combing ML with traditional CO algorithms, and divided them into three kinds: (a) end-to-end learning methods, which use ML to directly solve the problem; (b) ML-first methods, which apply ML to provide meaningful properties of optimization problems and guide the search direction for CO algorithms; (c) ML-alongside methods, which utilize ML during the iterative process of optimization algorithms. The end-to-end learning methods are suitable for real-time applications, while the latter two are still time-consuming due to the CO algorithms.

Recently, end-to-end machine learning methods have been explored on CO problems, especially on TSP. Vinyals et al. [35] proposed a pointer network model to tackle the Euclidean TSP with supervised learning based on sequence-to-sequence framework. The encoder and decoder are both constructed by recurrent neural networks, which make it possible to solve different input graph sizes. Nevertheless, this supervised learning model has the limitation of strong dependence on high-quality labels. To overcome the drawback, Bello et al. [4] used a reinforcement learning method to train the similar pointer network and set the tour length as a reward signal. To avoid the influence of input sequence on the model, Khalil et al. [18] employed a graph neural network to process the input data, and combined the reinforcement learning to address the problem. By modifying the pointer network with attention mechanism and reinforcement learning method, the solutions gained by Kool et al. [19] have been improved over recent heuristics for TSP. Besides, Ma et al. [22] introduced the graph pointer network trained by reinforcement learning which performed better than the pointer network [35], but could not dominate the attention model [19].

Most of the above literatures assumed all the parameters as deterministic values. However, uncertainty is ubiquitous and inevitable in real life. Powell [28] published a comprehensive review of the stochastic optimization. As mentioned, the stochastic problems can be solved either exactly [15] or approximately [40]. The former usually assumes the distribution functions to be additive such as gamma distribution and normal distribution, or can be decomposed into multiple additive functions. The latter includes sampling methods such as Monte Carlo sampling or direct online observations (also called data driven approach). However, the convergence of the Monte Carlo method is very slow: the ultimate accuracy cannot be improved faster than $O(1/\sqrt{N})$, where N

is the number of simulation samples or replications [20]. The computational cost will be prohibitively high if the problem is complex and requires high accuracy.

To speed up the stochastic optimization, some methods have been developed. For ranking and selection problems, Chen and Lee [8] proposed an optimal computing budget allocation method, which allocates the samples sequentially to optimize the selection quality under a simulation budget constraint. Besides, Bengio et al. [7] proposed a supervised learning method to find the representation scenario (RS) and transformed the stochastic problem to a deterministic one, which could obtain similar solution quality with much less computing time than general algorithms. However, the RS could not always exist and the generalization ability of the model is unsatisfactory sometimes. Although this method greatly reduces the computational time compared to sampling methods, it sacrifices accuracy to a certain extent.

As for TSP and PDP, uncertainty mainly lies in travel time, food preparation time, service time, and customer demand. For the dynamic PDP with stochastic food preparation time, Ulmer et al. [33] assumed that the time was gamma distributed and presented a cost function approximation with time buffers to solve the uncertainty. For the stochastic TSP with pickups and deliveries, Elgesem et al. [10] assumed that the travel time was independent normal distributed, and employed several exact methods based on Monte Carlo simulation to solve the problem. As for the green vehicle routing problem with stochastic costumer demands, Niu et al. [27] generated the mean demand according to a discrete uniform distribution and proposed a membrane-inspired multi-objective algorithm to solve the problem. The above literatures all assumed that the random variables obeyed independent and additive functions. However, this assumption may lose some information of real data distributions. In this paper, we assume the food preparation time as stochastic variables and propose the stochastic ORPP (SORPP). The discrete distribution functions of the food preparation time are predicted by an ML model trained by historic data from Meituan. The functions are very complex with long tail, multimodality, and without additivity. To our best knowledge, there are no other research works that consider stochastic variables with such kind of distributions.

From the literature review, it can be seen that the existing exact algorithms or meta-heuristics of related problems are inappropriate to solve the SORPP due to their unacceptable computational time. Although some heuristics can solve related problems quickly, they cannot guarantee the quality of the obtained solutions. Therefore, the core challenge for SORPP is how to obtain satisfactory solutions within a very short period of time. Hence, we use an end-to-end machine learning method to solve the proposed problem efficiently. As mentioned before, the existing research works of end-to-end learning can mainly be classified into two types: supervised learning [35] and reinforcement learning [4, 18, 19, 22]. The former is relatively easy to implement, but strongly depends on the label quality. In the case of high-quality labels, supervised learning can well imitate the "expert experience" and learn the optimization policy to generate satisfying solutions. The latter does not require labeling and complex feature engineering, but it is difficult to design appropriate reward/action/state functions. With approximated policy, the reinforcement learning model may fall into local optima easily. Besides, it usually costs much longer training time than supervised learning. In real-life situation of Meituan platform, each rider can only carry a small number of packages limited by the trunk capacity. If the computational time is not limited, the optimal (or approximate optimal) solutions of the problems on this scale are easy to find. That is, we can obtain plenty of high-quality labels of the problem. Therefore, we design an end-to-end deep learning model trained by supervised learning to solve the problem. The model is denoted as Meituan stochastic delivery network (MSDN).

Overall, the major contributions of this paper can be summarized as follows:

1. We propose the stochastic online route-planning problem for the first time which is formulated by a two-stage stochastic programming mathematical model.
2. We design an end-to-end deep learning method to solve the SORPP. In the encoder, the model produces the embeddings of all input features by specially designed network layers. In the decoder, the permutation sequence is predicted by long-short term memory (LSTM) cells with attention and masking mechanism.
3. We present problem-specific features to improve the performance of the model.
4. We adopt the IG algorithm based on Monte Carlo sampling to obtain high-quality labels for model training.
5. We conduct extensive experiments on the real-world data sets from Meituan. The results show the effectiveness and efficiency of the proposed model.

The remaining of the paper is organized as follows. The next section provides the description and formulation of the SORPP. The continuous section introduces the IG algorithm for labeling. And the following section presents the details of the proposed MSDN. Computational results and comparisons are reported in the consequent section. Finally, the paper is ended with some conclusions and future work in the last section.

Problem description

A problem instance of the SORPP comprises one rider and n orders, denoted as $W=\{w_1, w_2, \ldots, w_n\}$. Each order specifies a pickup point (the corresponding restaurant) and a delivery point (the corresponding customer). As shown in Fig. 2, the rider will start from the current position and pick up or deliver food according to the planned route where the food preparation time of unpicked orders is stochastic. At the scheduling time, some of the orders have already been picked up, so we only consider their delivery points. $P=\{1, \ldots, n_1\}$ is the set of pickup points and $D=\{n_1+1, \ldots, n_1+n\}$ is the set of the delivery points, where n_1 is the number of pickup points. In addition, an order is represented as a point pair (i, n_1+i), $i\in[1, n_1]$, or $(-1, n_1+i)$, $i\in[n_1+1, n]$, where -1 is the pickup point which has already been visited before.

In Meituan's situation, some basic constraints are given as follows.

1. Precedence constraints. The rider must pick up the food before deliver it.
2. Time window constraints. The rider must pick up the food after it has been prepared (hard time-window) and should try to deliver it before the promised time, also called ETA (soft time-window). Since the food preparation time is stochastic, all the values sampled from the corresponding probability mass functions (PMFs) should meet the time-window constraints.

The goal of the SORPP is to minimize the expected time cost, denoted as ETC. The problem can be modeled as a two-stage stochastic programming with the following notations.

Parameters:

v	The speed of the rider.
n	The number of orders.
n_1	The number of pickup points.
W	The set of all orders.
K	The set of scenarios.
N	The set of all points, $N=\{0, 1, \ldots, n_1+n\}$, where 0 is the starting point of the rider.
P	The set of pickup points, $P=\{1, \ldots, n_1\}$.
D	The set of delivery points, $D=\{n_1+1, \ldots, n_1+n\}$.
d_{ij}	The distance between point i and j, $i, j\in N$.
t_{ij}	The travel time between point i and j, $i, j\in N$.
e_i	The ETA of point i, $i\in D$.
ξr_i^k	The preparation time of point i in scenario k, $i\in P$, $k\in K$.
M	A sufficiently large positive number.
$\lvert * \rvert$	The number of elements in a certain set

Decision variables:

$x_{i,j}$	1, If the point j is assigned after i; 0, otherwise, i, $j\in N$.
u_i	The sequence of point i, $i\in N$. $u_i\in[0, n_1+n]$, $u_i\in\mathbb{Z}$.
a_i^k	The arrive time of point i in scenario k, $i\in N$, $k\in K$.
l_i^k	The leave time of point i in scenario k, $i\in N$, $k\in K$.
tw_i^k	The wait time of point i in scenario k, $i\in P$, $k\in K$.
to_i^k	The overtime of point i in scenario k, $i\in D$, $k\in K$

$$\min_{x,u} E_{\mathrm{TC}} = \sum_{i\in N} \sum_{j\in N-\{0\}} \frac{d_{i,j}x_{i,j}}{v} + \mathbb{Q}(x, \xi r) \tag{1}$$

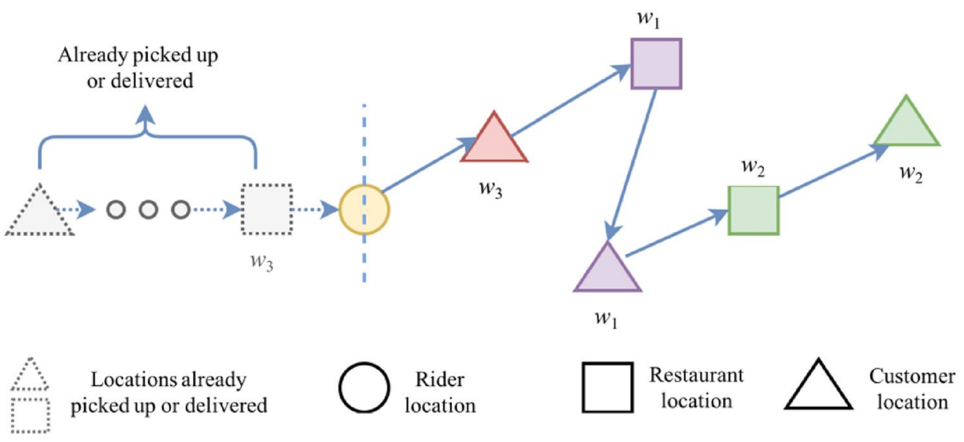

Fig. 2 An instance of the SORPP

s.t.

$$\sum_{j\in N-\{0\}} x_{i,j} = 1, \quad \forall i \in N \tag{2}$$

$$\sum_{j\in N-\{0\}} x_{j,i} = 1, \quad \forall i \in N - \{0\} \tag{3}$$

$$u_i \le M - 1 - (M-2)x_{0,i}, \quad \forall i \in N - \{0\} \tag{4}$$

$$u_i - u_j + (M-1)x_{i,j} + (M-3)x_{j,i} \le M - 2, \quad \forall i \in N, j \in N, j \ne i \tag{5}$$

$$u_i \le u_{i+n} - 1, \quad \forall i \in P \tag{6}$$

$$u_0 = 0 \tag{7}$$

$$x_{i,j} \in \{0,1\}, \quad \forall i \in N, j \in N - \{0\}, j \ne i \tag{8}$$

$$u_i \in \left[0, n_1 + n\right]N, \quad \forall i \in N, \tag{9}$$

where:

$$\mathbb{Q}(x, \xi r) = \min_{to,tw} \frac{1}{|K|} \sum_{k\in K} \left(\sum_{i\in D} \mathrm{to}_i^k + \sum_{i\in P} \mathrm{tw}_i^k \right) \tag{10}$$

s.t.

$$a_j^k - l_i^k - Mx_{i,j} \ge \frac{d_{i,j}}{v} x_{i,j} - M, \quad \forall i \in N, j \in N - \{0\}, j \ne i, \forall k \in K \tag{11}$$

$$a_j^k - l_i^k + Mx_{i,j} \le \frac{d_{i,j}}{v} x_{i,j} + M, \quad \forall i \in N, j \in N - \{0\}, j \ne i, \forall k \in K \tag{12}$$

$$l_i^k \ge a_i^k, \quad \forall i \in N, \forall k \in K \tag{13}$$

$$l_i^k \ge \xi r_i^k, \quad \forall i \in P, \forall k \in K \tag{14}$$

$$l_0^k = 0, \quad \forall k \in K \tag{15}$$

$$a_0^k = 0, \quad \forall k \in K \tag{16}$$

$$\mathrm{tw}_i^k \ge \xi r_i^k - a_i^k, \quad \forall i \in P, \forall k \in K \tag{17}$$

$$\mathrm{to}_i^k \ge l_i^k - e_i, \quad \forall k \in K \tag{18}$$

$$\mathrm{tw}_i^k \ge 0, \quad \forall i \in P, \forall k \in K \tag{19}$$

$$\mathrm{to}_i^k \ge 0, \quad \forall i \in D, \forall k \in K. \tag{20}$$

The objective function (1) is to minimize the expected time cost ETC which includes the rider traveling time, the expected waiting time, and overtime under stochastic situation. ***x*** and ***u*** are the first-stage decisions. **tw** and **to** are the second-stage decisions. Constraints (2) and (3) imply that each order can only be picked up once and delivered once, and the starting point will only be select once. Constraints (4) and (5) indicate the transformation of ***x*** and the sequence of the points ***u***. Constraints (6) ensure that the precedence relationships cannot be violated. Constraints (7)–(9) display the value ranges of the first-stage decision variables. The objective function (10) of the second stage is to minimize the expected waiting time and overtime under stochastic situation. Constraints (11) and (12) reveal that the arrive time of one point is equal to the leave time of the previous visited point plus the traveling time between the two in a certain scenario. Constraints (13) and (14) show that the leave time of each point is larger than the arrive time and the preparation time (for pickup points). Constraints (15) and (16) denote that the leave time and arrive time of the starting point are both 0. In addition, Constraints (17) define that ***tw*** is larger than the difference between the stochastic food preparation time and the arrive time for pickup points. Similarly, ***to*** is larger than the difference between the leave time and the ETA for delivery points as shown in Constraints (18). Constraints (19) and (20) demonstrate the value ranges of ***tw*** and ***to*** in each scenario.

The IG algorithm for labeling

As mentioned before, we employ supervised learning to train the model where the labels are obtained by the IG algorithm. The IG algorithm is of powerful exploitation capability and has been successfully applied to various scheduling problems [25]. As shown in Fig. 3, the main procedure of the IG algorithm includes initialization, destruction, reconstruction, and problem-specific local search.

Representation

A solution is represented by $1+n_1+n$ permutation sequence Π, including the starting point of the rider, the pickup and delivery points of all orders. For an example with $n_1=2$, $n=3$, as shown in Fig. 1, the permutation sequence is $\Pi=\{0, 5, 1, 3, 2, 4\}$, where the starting point is set as 0, and w_1, w_2, w_3 are associated with (1, 3), (2, 4), and (− 1, 5), respectively. The point − 1 is ignored in the problem.

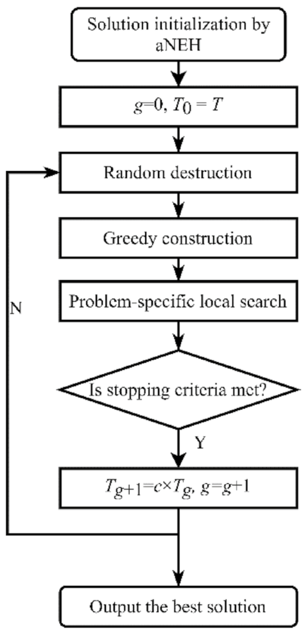
Solution initialization by aNEH
g=0, $T_0 = T$
Random destruction
Greedy construction
Problem-specific local search
Is stopping criteria met?
N
Y
$T_{g+1}=c\times T_g$, $g=g+1$
Output the best solution

Fig. 3 The framework of iterated greedy algorithm

Initialization

The initial route is generated by the adaptive Nawaz–Enscore–Ham (aNEH) heuristic [25]. The main steps of the aNEH are as follows.

Step 1: Set initial permutation as $\Pi_0 = \{0\}$.

Step 2: Rank n orders by their ETAs in an ascending order, $W = (w_{(1)}, w_{(2)}, \ldots, w_{(n)})$ where $w_{(i)}$ represents the ith order with the least ETAs. Let $i = 1$.

Step 3: Insert the pickup point of $w_{(i)}$ into all position of Π_0. Once assigned, insert the delivery point of $w_{(i)}$ into all positions after the position of the related pickup point. The permutation with minimal ETC will be reserved and replace Π_0. The ETC is calculated by Monte Carlo sampling.

Step 4: If $i \le n$, $i = i + 1$, go to Step 3; otherwise, output Π_0.

By this way, a solution with certain quality is generated.

Random destruction

In the destruction phase, α ($\alpha < n$) orders are randomly selected, with the removal of their pickup and delivery points from the permutation sequence. The chosen point pairs constitute a list, denoted as $L_S = \{\text{pair}_i, i \in [1, \alpha]\}$, and the remaining permutation is denoted as Π_R.

Greedy reconstruction

In the reconstruction phase, the L_S is shuffled at first to ensure sufficient randomicity. Then, the pickup point (if not − 1) and the delivery point of the order in L_S will be inserted into Π_R successionally. The partial solution with minimal ETC will be reserved greedily.

Problem-specific local search

To further improve the performance of the algorithm, a problem-specific local search is designed with following two neighborhood search operators.

Backward search

Find the delivery points with largest expected overtime and move them backward to an optimal position.

Forward search

Find the points with most sufficient time and move them forward to an optimal position.

Acceptance and stopping criteria

To avoid falling into local minimum, we employ the acceptance criteria of simulated algorithm. That is, we not only accept the solutions better than current one, but also worse solutions sometimes according to an acceptable probability. The probability is $p = e^{-\left(E_{\mathrm{TC}}' - E_{\mathrm{TC}}^{\mathrm{best}}\right)/T}$, where $T > 0$ is the current temperature, E_{TC}' is the objective function value of a certain solution, and $E_{\mathrm{TC}}^{\mathrm{best}}$ is the objective function value of the best solution obtained before. T is initialized with initial temperature T_0, and is updated as $T_{g+1} = c \times T_g$ at iteration g. $c \in (0, 1)$ is the cooling rate.

The algorithm will be finished if one of the following stopping criteria is met: the maximum number of iterations $g_{\max}$ has been reached; the best solution is not improved for t consecutive iterations.

MSDN for SORPP

We design an end-to-end deep learning model to solve the SORPP in an online fashion. The MSDN is composed of an encoder and a decoder, which is shown in Fig. 4. The encoder is used to produce embeddings of all input features, and the decoder can produce the sequence of the route points based on the output of the encoder.

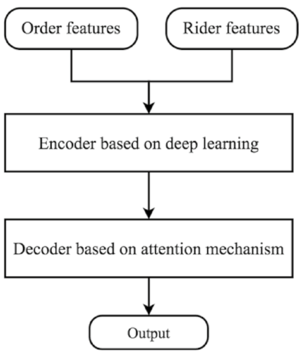
Order features
Rider features
Encoder based on deep learning
Decoder based on attention mechanism
Output

Fig. 4 The framework of the MSDN

Feature engineering

Feature engineering is to extract features from raw data and transform them into applicable format, which is a crucial part in machine learning. Suitable features can reduce the difficulty of modeling and improve the performance of output results [41]. However, it is challenging to design effective features which usually depend on expert knowledge about the optimization problem and statistics analysis on large quantity of data.

In our paper, basic features of the SORPP include the rider and order information as follows.

Rider-related features

The location (latitude and longitude) of starting point, the average speed during the last month, the number of carried orders, and the number of pickup points and delivery points.

Order-related features

The ETAs, the locations of the pickup and delivery points, respectively, and the PMFs of the food preparation time.

Although these features are enough to define an instance of the SORPP, they cannot well reveal the law of optimal solutions. For example, the order which is closer to the rider or is more urgent will possibly be visited with higher priority. Therefore, we design problem-specific features to describe the urgency of orders, the position relationship (including distance and time) between orders, as well as the position relationship between orders and the rider, as shown in Fig. 5. The urgency is represented by the remaining time of order delivery, defined as $\text{eta}_i - d_{0,i}/v$ for the orders already picked up, while $\text{eta}_i - (d_{i,i+n} + d_{0,i})/v$ otherwise. The distance and travel time between order points and the rider position are denoted as $d_{0,i}$ and $d_{0,i}/v$, $i \in N - \{0\}$, respectively, where $d_{0,i}$ is calculated by the longitude and latitude position information. Besides, the distance and travel time

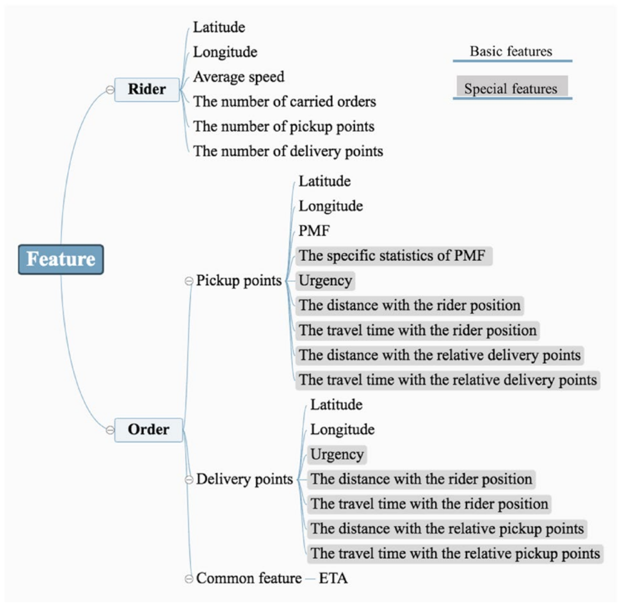
Feature
Rider
Latitude
Longitude
Average speed
The number of carried orders
The number of pickup points
The number of delivery points
Basic features
Special features
Order
Pickup points
Latitude
Longitude
PMF
The specific statistics of PMF
Urgency
The distance with the rider position
The travel time with the rider position
The distance with the relative delivery points
The travel time with the relative delivery points
Delivery points
Latitude
Longitude
Urgency
The distance with the rider position
The travel time with the rider position
The distance with the relative pickup points
The travel time with the relative pickup points
Common feature
ETA

Fig. 5 All features of the SORPP

between the pickup and delivery points of each order are $d_{i,i+n_1}$, $d_{i,i+n_1}/v, i \in [1, n_1]$, respectively, for orders which have not been picked up and $d_{0,i+n_1}$, $d_{0,i+n_1}/v, i \in [n_1 + 1, n]$ otherwise. Besides, the specific statistics of PMF is composed of the corresponding mean, sum, medium, maximum, minimum values, and standard deviation. By introducing these problem-specific features, the performance of the model is further improved.

We pre-process the data by cleaning missing data and replacing anomalous data with average values. Besides, all the continuous features are normalized first according to the average values and standard deviations of the training set.

Encoder

The encoder is used to transform the input into an intermediate embedding. As shown in Fig. 6, we deal with different types of features separately and then concatenate them to a final embedding. The encoder component includes three parts: rider embedding of rider features, order embedding of order features, and location embedding of the location matrix which consists of all the location information extracted from the rider and all orders.

Location embedding

Because the latitudes and longitudes are nonnumeric information, we adapt the GPS embedding (or cell coding) [21] to represent them. The main idea of GPS embedding is to divide the area into small grids with corresponding geographical information according to certain rules. Compared

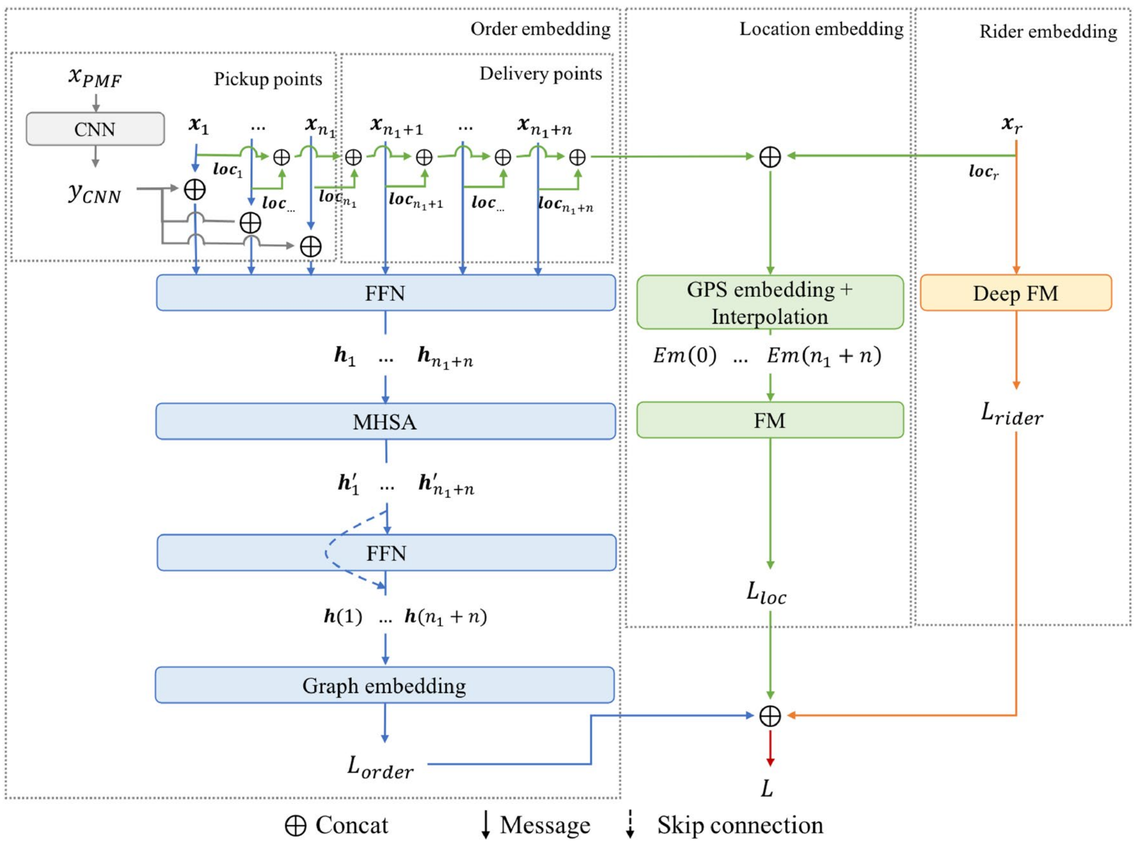

Fig. 6 The encoder of the MSDN

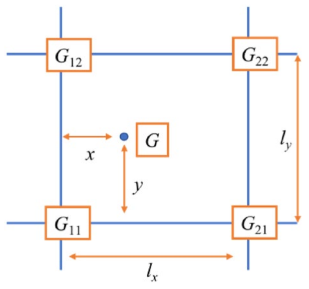

Fig. 7 An instance of grid code

to geohash method [26] or one-hot representation [6], the GPS embedding can depict GPS information more accurately based on distance weights. Therefore, it can avoid the sudden change of neighborhood location and make the embedding smoother. The main steps of the GPS embedding are as follows.

Step 1: Map the latitudes and longitudes into grids of 300×300 m^2.

Step 2: Reserve the grids which are not out of vocabulary. Since some of the grids have one or multiple GPS points, while others may have none, we only reserve the grids with multiple GPS points according to a threshold (set as 300) to guarantee the continuity of a route.

Step 3: Transform the retained grids into a d_1-dimensional embedding denoted as Em(*).

Step 4: Calculate the embedding of each location based on bilinear interpolation of 4 nearest vertices of the grid. For an example, as shown in Fig. 7, G is a certain location which is concatenated with 4 vertices, denoted as G_{ij} $i, j = \{1, 2\}$. The embedding of G is calculated as follows:

$$\mathrm{Em}(G) = \frac{(l_x - x)(l_y - y)}{l_x \times l_y}\mathrm{Em}(G_{11}) + \frac{x(l_y - y)}{l_x \times l_y}\mathrm{Em}(G_{21}) + \frac{(l_x - x)y}{l_x \times l_y}\mathrm{Em}(G_{12}) + \frac{xy}{l_x \times l_y}\mathrm{Em}(G_{22}). \tag{21}$$

To better reflect the location relationship between the points, we extract the location features from all the points and form a global location embedding with $(1 + n + n_1, d_1)$ size by above method, denoted as Em_{loc}. Then, a factorization-machine (FM) [30] layer is employed to fuse the features. The FM can find the correlation of features by combing and overlapping them. The formulation of FM is as follows:

$$y_{\mathrm{FM}} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=0}^{d} \sum_{j=i+1}^{d} \langle \boldsymbol{V}_i, \boldsymbol{V}_j \rangle x_i \cdot x_j, \tag{22}$$

where $\boldsymbol{w} \in \mathcal{R}^d$, $\boldsymbol{V}_i \in \mathcal{R}^k$ are weight vectors, d is the number of features, k is the given embedding length, and x_i is the value of ith feature. The addition unit $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ reflects the importance of order-1 features and the inner product units represent the impact of order-2 feature interaction. In the location embedding, $\boldsymbol{x} = \mathrm{Em}_{\mathrm{loc}}$, $d = n + n_1$, and $k = d_1$. The FM acts on the first axis and outputs a vector with the size of d_1 which is denoted as $\boldsymbol{L}_{\mathrm{loc}}$.

By this method, all the locations can be mapped into a reasonable embedding which implies the spatial distance relationship between them.

Rider embedding

Rider features include one continuous feature (rider speed) and five discrete features. The latitude and longitude of the starting point is disposed by above method. To avoid information loss during the training process, we increase the dimension of other numeric features by transforming them into a d_2-dimensional learnable embedding, denoted as e_r. The deep FM [14] is adopted to further process the rider features thanks to its effectiveness in combinatory features. Deep FM consists of an FM and a deep neutral network (DNN). The former is used to learn the linear and pairwise interactions between features and the latter is used to learn high-order feature interactions. The FM and DNN component shares the same input features. Set $\boldsymbol{x}^{(0)} = [\mathrm{Em}(0), \boldsymbol{e}_r]$ as the initial input. $\boldsymbol{y}_{\mathrm{FM}}$ can be calculated as Eq. (22) in the FM component. In addition, in the DNN component, $\boldsymbol{x}$ is updated by:

$$\boldsymbol{x}^{(i+1)} = \sigma(\boldsymbol{W}^i \boldsymbol{x}^i + \boldsymbol{b}^i), \tag{23}$$

where i is the layer depth and σ is an activation function which is set as ReLU function [11] in this paper. $\boldsymbol{x}^i$, $\boldsymbol{W}^i$ and $\boldsymbol{b}^i$ are the output, model weight, and bias of the ith layer, respectively. Then, a dense vector is generated and fed into the activation function as follows:

$$\boldsymbol{y}_{\mathrm{DNN}} = \sigma(\boldsymbol{W}^{|H|+1} \boldsymbol{x}^H + \boldsymbol{b}^{|H|+1}), \tag{24}$$

where $|H|$ is the number of hidden layers. The output rider embedding is denoted as $\boldsymbol{L}_{\mathrm{rider}} = [\boldsymbol{y}_{\mathrm{FM}}, \boldsymbol{y}_{\mathrm{DNN}}]$.

Order embedding

The order embedding is mostly important in the encoder component based on attention mechanism, which enables it to extract important information effectively. The order embedding contains the basic information of all the order points which affect the generation of the permutation sequence.

Similarly, the order locations are disposed by GPS embedding, and turned into a d_1-dimensional embedding for each point. The other numeric features are transformed into a d_3-dimensional embedding for each pickup point or delivery point. The initial order embeddings are generated by concatenation, denoted as $\boldsymbol{x}_i$, $i \in [1, n_f]$.

It is typical to generate a large number of scenarios to approximately describe complex PMFs. However, the computational time of this method is unaffordable for online system. Since the PMFs can be regarded as a 2-D image-like matrix and convolution neural network is widely employed for image classification and compression [32], we adopt the CNN with one hidden layer to pre-process them and obtain global information of food preparation time. The filter size is 3×3, and the output vector is concatenated with the embedding of pickup point features, while the delivery points remain unchanged as follows:

$$\boldsymbol{x}_i = \begin{cases} [\boldsymbol{x}_i, \boldsymbol{y}_{\mathrm{CNN}}], & i \in P \\ \boldsymbol{x}_i, & i \in D. \end{cases} \tag{25}$$

By this way, the features of pickup and delivery points are turned into learnable embeddings, respectively. And a feed-forward network (FFN) [31] is employed to match their size as follows:

$$\boldsymbol{h}_i = \mathrm{FFN}(\boldsymbol{x}_i), \quad i \in N - \{0\}. \tag{26}$$

After the above pre-processing, we employ the similar encoder used in attention model by [19] to extract valuable information of order features for its effectiveness in TSP. The point embeddings are updated by a multi-head self-attention (MHSA) and a node-wise FFN. Each layer adds a skip-connection [16] and batch normalization (BN) [2]. The specific structure can be referred to [19]:

$$\boldsymbol{h}_i^{'} = \mathrm{BN}(\boldsymbol{h}_i + \mathrm{MHSA}_i(\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_{n_f})), \quad i \in N - \{0\} \tag{27}$$

$$\boldsymbol{h}(i) = \mathrm{BN}(\boldsymbol{h}_i^{'} + \mathrm{FFN}(\boldsymbol{h}_i^{'})), \quad i \in N - \{0\}. \tag{28}$$

Then, a graph embedding is calculated as the mean of the final node embeddings, denoted as $\boldsymbol{L}_{\mathrm{order}} = \frac{1}{n_f} \sum_{i=1}^{n_f} \boldsymbol{h}(i)$. The final embedding in encoder consists of the graph embedding of order features, the rider embedding, and the location embedding as follows. It is used to compose the input of the decoder together with point embeddings:

$$\boldsymbol{L} = [\boldsymbol{L}_{\mathrm{loc}}, \boldsymbol{L}_{\mathrm{rider}}, \boldsymbol{L}_{\mathrm{order}}]. \tag{29}$$

Decoder

The decoder is used to generate the permutation sequence based on attention mechanism similar to the pointer networks [34] and problem-specific masking mechanism. The former allows the decoder to quickly judge the importance of each points and the latter guarantees the feasibility of generated solutions. The decoder consists of a LSTM cell and a softmax layer. At each timestep t in $\{1, n_1+n\}$, the LSTM cell will output a point vector $g_{o,t}$ based on the point embeddings from the encoder. The implementation of the LSTM is shown in Eqs. (30)–(35), where $g_{f,t}$, $g_{i,t}$, $g_{o,t}$, $g_{c',t}$, $g_{c,t}$, and H_t are the output of forget gate, input gate, exposure gate, new memory cell state, final memory cell state, and hidden state, respectively. $W^{(*)}$ and $b^{(*)}$ are the weights and bias for each gate or state, and are shared among all cells at each timestep. Besides, $h_{c,t}$ is the decoder context at time t, which is composed of the point embedding and final embedding from encoder. Since the route starts at the location of the rider, the $h_{c,t}$ is set as Eq. (36):

$$g_{f,t} = \sigma(W^{(f)} \cdot [H_{t-1}, h_{c,t}] + b^{(f)}) \tag{30}$$

$$g_{i,t} = \sigma(W^{(i)} \cdot [H_{t-1}, h_{c,t}] + b^{(i)}) \tag{31}$$

$$g_{c',t} = \tanh(W^{(c)} \cdot [H_{t-1}, h_{c,t}] + b^{(c)}) \tag{32}$$

$$g_{c,t} = g_{f,t} * g_{c,t-1} + g_{i,t} * g_{c',t} \tag{33}$$

$$g_{o,t} = \sigma(W^{(o)} \cdot [H_{t-1}, h_{c,t}] + b^{(o)}) \tag{34}$$

$$H_t = g_{o,t} * \tanh g_{c,t} \tag{35}$$

$$h_{c,t} = \begin{cases} [L, L_{\text{rider}}], & t = 1 \\ [L, h(\pi_{t-1})], & t > 1. \end{cases} \tag{36}$$

To ensure the feasibility of the solution, the candidate points must satisfy the constraints of the SORPP. To meet the precedence constraint, the points will be masked (set the pointer vector $u_{i,t} = -\infty$) if the corresponding food has not been picked up by the rider. Likewise, to guarantee each point only appears once, the points that have already been visited will also be masked. By this way, the pointer vector $u_{i,t}$ is defined as:

$$u_{i,t} = \begin{cases} -\infty, & \text{if } i \neq \pi_{t'}, \forall t' < t \\ -\infty, & \text{if } \nexists (i-n) = \pi_{t'}, \forall t' < t, \ i \in D \\ v^{\mathrm{T}}(W_1 L_{\text{order}} + W_2 H_t) & \text{otherwise} \end{cases}, \tag{37}$$

where W_1 and W_2 are trainable matrices. Then, a distribution over the next candidate points is generated by passing the vector into the softmax layer as follows. The one with the largest probability will be chosen as the next point:

$$p_{i,t} = \text{softmax}(u_{i,t}). \tag{38}$$

An example is shown in Fig. 8 which explains how to generate a feasible permutation sequence. Two orders are considered and represented by (1, 3) and (2, 4), respectively. The decoder takes as input the graph embedding and point embeddings. Note that the point 3 and 4 are masked at $t=1$ because the corresponding orders have not been picked up and the point 1, 2, and 3 are masked at $t=4$, because they have been visited before. The permutation sequence $\Pi = \{0, 2, 1, 3, 4\}$ is constructed at the end.

Loss function

Since the decoder can be regarded as a multi-classify model, the cross-entropy loss function as follows is employed in our paper, which is widely used in multi-classify problems:

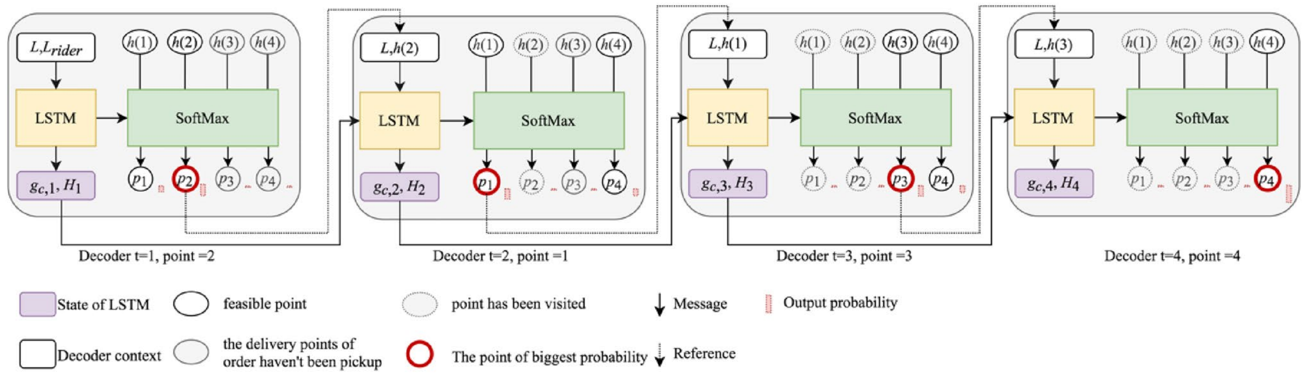

Fig. 8 An example of the decoder for the SORPP

$$L(\pi_L|N;\theta) = -\frac{1}{n_f}\sum_{i\in N-\{0\}} \pi_i \log\left(p_i;\theta\right), \tag{39}$$

where π_L is the permutation sequence of label, π_i is the ith point in π_L, p_i is the predicted possibility of π_i at $t=i$, and θ represents all the parameters of the model. By this way, the distance of the predicted route and the label can be estimated. Given a training pair (N, π_L), the parameters of the MSDN are learnt by maximizing the loss function for the training set, that is:

$$\theta^* = \arg\max_{\theta} \sum_{\pi_L,N} L(\pi_L|N;\theta). \tag{40}$$

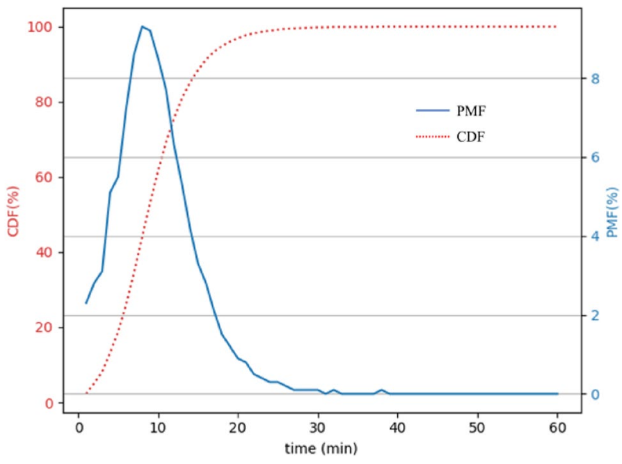

Fig. 10 An instance of the CDF and PMF of the food preparation time

Computational results

Experimental settings

In this section, numerical experiments are conducted to evaluate the performance of the MSDN. The data sets are sampled from real historical data across China in Meituan platform. Two kinds of training and validation sets are generated, denoted as TS1 and TS2, respectively, where TS1 obeys the distribution of real data and TS2 is equalized according to n. Besides, the test set is shared by all models and comparison algorithms which will be introduced later. The data distribution is shown in Fig. 9, where the numbers around the pie chart represent the order number n of a route. The PMFs are obtained from extensive history data and are assumed to be known in this paper. Figure 10 shows an example of a PMF and its cumulative distribution function (CDF) correspondingly. All the parameters are empirically set, as shown in Table 1. The model is trained in Python 2 on Meituan servers and saved as a checkpoint file which is used to invoke the model during the test experiments under Mac OS. Besides, all the comparative algorithms (introduced later in the paper) are coded in Java SE8. The experiments are run on a MacBook Pro with 2.2 GHz processors/16 GB RAM under Mac OS.

For the SORPP, the performances of the algorithms are evaluated in terms of computational time and solution quality. In this paper, we employ the average CPU time to measure the computational time and the following two metrics to evaluate the solution quality.

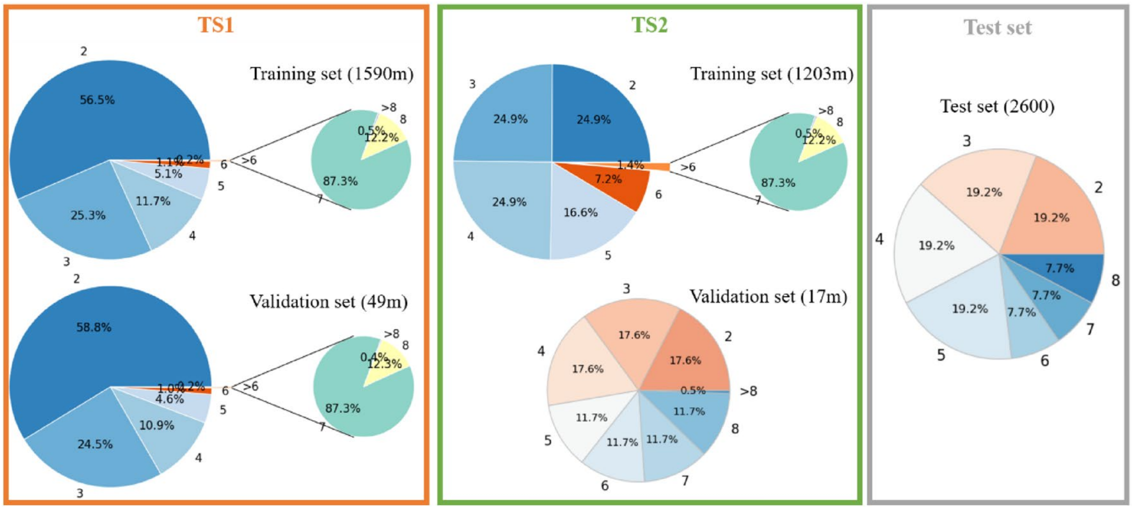

Fig. 9 The distribution of different data sets

Table 1 Parameter values

	Parameter	Value
Hyper parameters	Epoch	8
	Batch size	256
	Training steps (per epoch)	2500
	Optimizer	Adam
	Learning rate	0.003
	Learning rate decay	0.95
Parameters of the MSDN	d_1	128
	d_2	128
	d_3	128
	$\|H\|$	3
Parameters of the IG algorithm	α	3
	T_0	500
	$g_{\max}$	200
	c	0.95
	Sampling number	10,000

Route consistency (RC)

$$\mathrm{RC} = \frac{S_m}{\mathrm{len}_s}, \tag{41}$$

where len_s is sequence length and S_m is the length of longest common prefix between two permutation sequences. In this paper, 0 is excluded, since all the sequences start from 0. For example, supposing two solutions $A = \{0, 1, 2, 3, 4\}$ and $B = \{0, 1, 3, 4, 2\}$. S_m is 1 (the length of $\{1\}$), and len_s is 4 after excluding 0. Therefore, the RC of A and B equals to $1/4 = 0.25$. If the first nonzero number of two solutions is not the same, the RC = 0 despite how similar the successive points can be. The RC will be higher if two solutions have longer common prefix. It can be used to measure the learning ability of models by evaluating the similarity between the solutions learned by the model and the labels.

Relative percentage deviation (RPD)

$$\mathrm{RPD} = \frac{\mathrm{alg} - \mathrm{lab}}{\mathrm{lab}} \times 100, \tag{42}$$

where alg is the solution obtained by a certain algorithm (or model), and lab is the label obtained by the IG algorithm in this paper. Different from the RC, the RPD is used to evaluate the solution quality of the algorithms and model. The algorithm performs better with a smaller RPD.

Effectiveness of special designs

In this section, we verify the effectiveness of several special designs, including data equalization, the design of the problem-specific features, and the utilization of the CNN. To demonstrate the effect of the problem-specific features, we compare the MSDN using basic features (*_BF) to the one with special features (*_SF). Besides, the model with (*_CNN) or without CNN layer (*_nCNN) are also tested to illustrate the influence of CNN layer. To show the impact of data equalization, TS1 and TS2 are used to train the model, respectively, denoted as *_TS1 and *_TS2. The results are shown in Tables 2 and 3.

From Tables 2 and 3, it can be seen that the MSDN_CNN_SF_TS2 performs best on average, which indicates the superiority of this model. Besides, the MSDN_CNN_SF_TS2 is better than MSDN_nCNN_SF_TS2 on most instance, which shows the effectiveness of the CNN layer on the information extract of PMFs. As for MSDN_CNN_SF_TS1, the model trained by TS1 is superior on instances with small n especially when $n = 2$, but is inferior with large n. Similarly, although MSDN_CNN_SF_TS2 can perform better on average when we equalize the training data, it also loses accuracy on the instances with small n. By comparing these two models, we can conclude that models with different training sets cannot always perform

Table 2 Average RC of different models

n	MSDN_CNN_SF_TS2	MSDN_nCNN_SF_TS2	MSDN_CNN_SF_TS1	MSDN_CNN_BF_TS2
2	**0.910**	0.906	**0.910**	0.747
3	0.807	0.796	**0.822**	0.592
4	**0.616**	0.608	0.597	0.390
5	**0.596**	0.584	0.577	0.346
6	**0.507**	0.493	0.475	0.357
7	**0.503**	0.491	0.493	0.348
8	**0.531**	0.514	0.518	0.307
Average	**0.613**	0.604	0.607	0.437

The best results are in bold

Table 3 Average RPD of different models

n	MSDN_CNN_SF_TS2	MSDN_nCNN_SF_TS2	MSDN_CNN_SF_TS1	MSDN_CNN_BF_TS2
2	0.08	0.10	**0.00**	3.87
3	0.19	0.49	**0.13**	7.04
4	**1.73**	2.54	2.20	12.02
5	1.57	**1.48**	1.80	13.56
6	**1.40**	2.16	2.08	8.27
7	**1.28**	1.39	1.35	10.83
8	**2.08**	2.31	3.37	13.22
Average	**1.05**	1.34	1.32	9.50

The best results are in bold

best on every instance, which lends support to no free lunch theorem [38] and shows the important influence of the data distribution on the model performance. Besides, the MSDN_CNN_BF_TS2 is obviously worse than others. The main reason is that the problem-specific features can provide more information for the model to learn the implicit relationship.

In general, we can draw the following conclusions: the CNN layer is effective to extract the information of PMFs; the problem-specific features can significantly improve the model performance; The distribution of training data will impact the model performance on instances of different scales.

Comparisons with other algorithms

To the best of our knowledge, there is little works carried out on SORPP. Therefore, we apply some algorithms which are commonly used in TSP or PDP as comparative algorithms. Due to the immediacy of the online problem, only heuristics or meta-heuristics are adopted as follows. The first four heuristics belong to constructive algorithms and the last three belong to (meta-)heuristics with iterative process, called iterative algorithm for convenience.

Random generation (RG)

The route is constructed randomly. If the solution is unfeasible, we will repair it by swapping the position of the illegal pickup point and its corresponding delivery point.

Earliest ETA first (EEF)

The order with earliest ETA will be inserted into the permutation sequence first. Its pickup and delivery points are assigned in order.

Most urgent first (MUF)

The most urgent order will be inserted into the permutation sequence first. Its pickup and delivery points are assigned consecutively. The urgency is defined similarly as in “Feature engineering”.

Nearest first (NF)

The points are sorted by the distance to the rider starting position. And the nearest point will be assigned first. If the solution is unfeasible, the illegal delivery point will be inserted right after its corresponding pickup point.

aNEH

Defined in “Initialization”.

IG with random initialization (IG_RG)

The solutions are initialized by RG and then improved by the destruction and reconstruction operators described in “The IG algorithm for labeling”. Due to time-limitation, these operators are only performed once.

IG with NF initialization (IG_NF)

It is the same as IG_RG except solutions are initialized by NF.

We use Monte Carlo sampling to dispose the uncertainty in these algorithms. To determine the sampling times, we compare the computational time and the fluctuation of the ETC under different sampling times, represented by $s = \{1, 10, 100, 1000, 10{,}000, 100{,}000\}$. Denoting the evaluation of a label under a certain s as a case, there are $6 \times 2600 = 15{,}600$ cases in total. To obtain credible results, each case runs 20 times to calculate the average computational time. Besides, six cases of the instance 1 under different s run 1000 times, respectively, to evaluate the fluctuation.

The results are shown in Table 4 and Fig. 11. Table 4 shows the average computational time of all the cases grouped by n. The computational time multiplies with the increasement of s, which shows the positivity between them. Figure 11 is a boxplot of the ETC on a certain permutation

Table 4 The average computational time under different sample times (ms)

n	100,000	10,000	1000	100	10	1
2	74.031	7.457	0.752	0.077	0.0075	0.00086
3	103.763	10.417	1.047	0.105	0.0124	0.0016
4	129.077	12.832	1.289	0.138	0.0144	0.0022
5	153.442	15.443	1.554	0.158	0.0178	0.0022
6	174.449	17.222	1.727	0.18	0.02	0.0024
7	171.149	17.219	1.722	0.179	0.0184	0.0034
8	155.529	15.655	1.595	0.163	0.0164	0.0016
Average	127.070	12.728	1.281	0.132	0.014	0.002

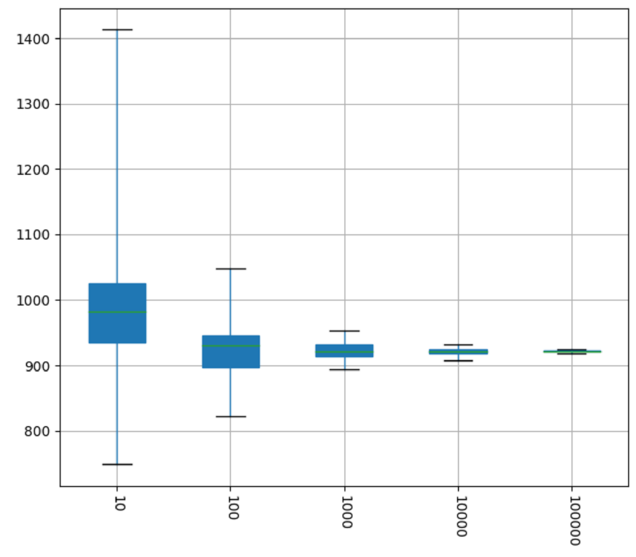

Fig. 11 The boxplot of the ETC of a certain permutation sequence

sequence (the label of instance 1). From the figure, it can be seen that with the decline of s, the fluctuation increases rapidly and the deviation of expectation enlarges promptly. By balancing the precision and elapsed time, we set the sampling times as 10,000 for the compared algorithms.

The comparison results of the model and the constructive algorithms are listed in Table 5, where the MSDN represents the MSDN_CNN_SF_TS2 for convenience. It can be seen that the computational time is similar between the model and the constructive algorithms. That is because most time of these methods is spent on solution evaluation by sampling method while constructing or predicting the sequence usually costs less than 1 ms. Although the consumed time is similar, the model performs much better than the constructive algorithms.

Table 6 shows the comparison results of the MSDN and the iterative algorithms. It can be seen that the average RPDs of the MSDN are superior than IG_RG and IG_NF on all instances and are better than aNEH on some instances. For instances with $n=2$ or 3, all these methods perform well, because it is easy for iterative algorithms or models to find the optimal solution. With the increasement of the point number in a route, the difficulty in solving the problem increases, which enlarges the average RPD. When $n=8$, the model performs slightly worse than aNEH mainly because of the lack of similar data in the training set. Although the average RPDs of the MSDN are not always the best, it is satisfying with relatively short computational time than the compared iterative algorithms, almost one-fiftieth of them. Therefore, the MSDN is more appropriate to be employed online than the compared iterative algorithms.

In conclusion, the MSDN is more effective and efficient to solve the proposed problem.

Table 5 Results of the constructive algorithms

n	Average RPD (%)					Time (ms)				
	RG	EEF	MUF	NF	MSDN	RG	EEF	MUF	NF	MSDN
2	17.66	14.68	15.71	3.58	**0.08**	8.19	8.17	8.50	8.16	**8.06**
3	48.25	36.74	38.86	10.04	**0.19**	11.48	10.97	12.61	11.21	**11.02**
4	88.79	40.53	44.51	23.26	**1.73**	14.27	14.15	14.37	13.64	**13.43**
5	123.78	54.56	59.56	30.44	**1.57**	17.46	17.17	21.76	17.18	**16.04**
6	161.39	48.47	50.46	46.09	**1.40**	19.41	18.82	29.99	18.08	**17.82**
7	215.08	56.74	59.73	71.36	**1.28**	19.02	18.49	18.35	18.31	**17.82**
8	216.17	38.82	41.50	70.48	**2.08**	20.62	19.47	18.89	19.17	**16.26**
Average	99.14	39.26	42.18	27.40	**1.05**	14.43	14.07	16.18	13.93	**13.33**

The best results are in bold

Table 6 Results of the iterative algorithms

n	Average RPD (%)				Time (ms)			
	aNEH	IG_RG	IG_NF	MSDN	aNEH	IG_RG	IG_NF	MSDN
2	0.10	0.10	0.10	**0.08**	55.03	62.13	55.03	**8.06**
3	0.42	6.35	1.46	**0.19**	161.12	162.95	161.12	**11.02**
4	**0.94**	16.94	5.01	1.73	445.49	424.43	445.49	**13.43**
5	**1.14**	25.93	7.34	1.57	851.02	924.69	851.02	**16.04**
6	1.95	44.58	14.28	**1.40**	1218.38	1472.89	1218.38	**17.82**
7	1.37	50.22	18.62	**1.28**	1289.88	1301.94	1289.88	**17.82**
8	**1.91**	63.47	18.57	2.08	1148.83	1138.54	1148.83	**16.26**
Average	**0.90**	21.66	6.64	1.05	572.21	603.76	572.21	**13.33**

The best results are in bold

Conclusions

In this paper, we addressed the stochastic online route-planning problem with the minimization of expectation time cost for the first time and established a two-stage stochastic programming mathematical model. It is a complex problem with immediacy requirement, uncertainty, precedence constraints, time-window constraints, and so on. To solve the problem in a very short time, we designed an end-to-end deep learning method and employed different network layers to tackle problem-specific features with different formats. According to the circumstances of real-world delivery, the model was trained by supervised learning to study the optimization policy under complex constraints, where the labels were obtained by iterated greedy algorithm. The experimental results showed that the MSDN is effective and efficient to solve the addressed problem on real-world data sets. Therefore, this research work can provide effective intelligent technique for complex online food delivery system.

In our future work, we will generalize the problem to large scale and propose the models with better generalization ability. In addition, it is interesting to solve the online route-planning problem with other types of uncertainties and to generalize the problems with more objectives including other robustness criteria.

Acknowledgements This study is supported by the National Science Fund for Distinguished Young Scholars of China (no. 61525304) and the National Natural Science Foundation of China (no. 61873328), and Meituan.

Compliance with ethical standards

Conflict of interest The authors declare that we have no conflict of interest.

References

1. Applegate DL, Bixby RE, Chvatal V, Cook WJ (2006) The traveling salesman problem: a computational study. Princeton University Press, Princeton
2. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv preprint http://arxiv.org/abs/1607.06450
3. Baker EK (1983) An exact algorithm for the time-constrained traveling salesman problem. Oper Res 31(5):938–945
4. Bello I, Pham H, Le QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. In: 2017 International conference on learning representations (ICLR). arXiv preprint http://arxiv.org/abs/1611.09940
5. Bengio Y, Andrea L, Antoine P (2018) Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv preprint http://arxiv.org/abs/1811.06128
6. Bengio Y, Courville A, Vincent P (2014) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828
7. Bengio Y, Frejinger E, Lodi A, Patel R, Sankaranarayanan S (2019) A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs. arXiv preprint http://arxiv.org/abs/1912.08112
8. Chen C, Lee LH (2011) Stochastic simulation optimization: an optimal computing budget allocation. World Scientific Publishing, Singapore
9. Edelkamp S, Gath M (2014) Solving single vehicle pickup and delivery problems with time windows and capacity constraints using nested Monte-Carlo search. In: Proceedings of the 6th international conference on agents and artificial intelligence (ICAART), pp 22–33
10. Elgesem AS, Skogen ES, Wang X, Fagerholt K (2018) A traveling salesman problem with pickups and deliveries and stochastic travel times: an application from chemical shipping. Eur J Oper Res 269:844–859
11. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the 14th international conference on artificial intelligence and statistics (AISTATS), pp 315–323
12. Grefenstette J, Gopal R, Rosmaita B, Van Gucht D (1985) Genetic algorithms for the traveling salesman problem. In: Proceedings of the 1st international conference on genetic algorithms and their applications, pp 160–168
13. Gülcü Ş, Mahi M, Baykan ÖK, Kodaz H (2018) A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem. Soft Comput 22:1669–1685
14. Guo H, Tang R, Ye Y, Li Z, He X (2017) DeepFM: a factorization-machine based neural network for CTR prediction. In: Proceedings of the 26th international joint conference on artificial intelligence (IJCAI). arXiv preprint http://arxiv.org/abs/1703.04247
15. Gupta S, Garg H, Chaudhary S (2020) Parameter estimation and optimization of multi-objective capacitated stochastic transportation problem for gamma distribution. Complex Intell Syst 6:651–667
16. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 770–778
17. Hosny MI, Mumford CL (2007) Single vehicle pickup and delivery with time windows: made to measure genetic encoding and operators. In: Proceedings of the 9th conference companion on genetic and evolutionary computation (GECCO), pp 2489–2496
18. Khalil E, Dai H, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. Adv Neural Inf Process Syst (NIPS) 30:6348–6358
19. Kool W, van Hoof H, Welling M (2019) Attention, learn to solve routing problems! In: 7th International conference on learning representations (ICLR). arXiv preprint http://arxiv.org/abs/1803.08475
20. Kushner HJ, Clark DS (2012) Stochastic approximation methods for constrained and unconstrained systems. Springer Science & Business Media, Berlin

21. Li X, Zhao K, Cong G, Jensen CS, Wei W (2018) Deep representation learning for trajectory similarity computation. In: 2018 IEEE 34th international conference on data engineering (ICDE), pp 617–628
22. Ma Q, Ge S, He D, Thaker D, Drori I (2019) Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. arXiv preprint http://arxiv.org/abs/1911.04936
23. Meituan (2020) Announcement of the results for the three months ended June 30, 2020. https://www1.hkexnews.hk/listedco/listconews/sehk/2020/0821/2020082100373.pdf. Accessed 10 Dec 2020
24. Morgan Stanley Research (2017) Is online food delivery about to get 'amazoned'? https://www.morganstanley.com/ideas/online-food-delivery-market-expands/. Accessed 10 Dec 2020
25. Nawaz M, Enscore EE Jr, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11:91–95
26. Niemeyer G (2016) Geohash. http://geohash.org. Accessed 10 Dec 2020
27. Niu Y, Zhang Y, Cao Z, Gao K, Xiao J, Song W, Zhang F (2021) MIMOA: a membrane-inspired multi-objective algorithm for green vehicle routing problem with stochastic demands. Swarm Evol Comput 60:100767
28. Powell WB (2019) A unified framework for stochastic optimization. Eur J Oper Res 275:795–821
29. Reinelt G (1991) TSPLIB—a traveling salesman problem library. Informs J Comput 3:376–384
30. Rendle S (2010) Factorization machines. In: 2010 IEEE international conference on data mining (ICDM), pp 995–1000
31. Sanger TD (1989) Optimal unsupervised learning in a single-layer linear feedforward neural network. Neural Netw 2:459–473
32. Traore BB, Kamsu-Foguem B, Tangara F (2018) Deep convolution neural network for image recognition. Ecol Inform 48:257–268
33. Ulmer MW, Thomas BW, Campbell AM, Woyak N (2020) The restaurant meal delivery problem: dynamic pickup and delivery with deadlines and random ready times. Transp Sci. https://doi.org/10.1287/trsc.2020.1000
34. Verhoeven M, Aarts EH, Swinkels P (1995) A parallel 2-opt algorithm for the traveling salesman problem. Future Gener Comput Syst 11:175–182
35. Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. In: 2015 Neural information processing systems (NIPS), pp 2692–2700.
36. Wang X, Wang L, Wang S, Chen J, Wu C (2021) An XGBoost-enhanced fast constructive algorithm for food delivery route planning problem. Comput Ind Eng 152:107029
37. Wang X, Wang S, Wang L et al (2020) An effective iterated greedy algorithm for online route planning problem. In: 2020 IEEE Congress on evolutionary computation (CEC), p 9185864.
38. Wolpert DH, Macready WG (1995) No free lunch theorems for search. Technical Report SFI-TR-95–02–010, Santa Fe Institute
39. Wrulich M, Hirschberg C, Rajko A, Schumacher T (2016) The changing market for food delivery. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-changing-market-for-food-delivery. Accessed 10 Dec 2020
40. Wu CC, Lin WC, Zhang XG, Bai DY, Tsai YW, Ren T, Cheng SR (2020) Cloud theory-based simulated annealing for a single-machine past sequence setup scheduling with scenario-dependent processing times. Complex Intell Syst. https://doi.org/10.1007/s40747-020-00196-7
41. Zheng A, Casari A (2018) Feature engineering for machine learning: principles and techniques for data scientists. O'Reilly Media, Inc., Newton