Solving Dynamic Traveling Salesman Problems With Deep Reinforcement Learning

Zizhen Zhang, Hong Liu, MengChu Zhou, *Fellow, IEEE*, and Jiahai Wang, *Senior Member, IEEE*

***Abstract*— A traveling salesman problem (TSP) is a well-known NP-complete problem. Traditional TSP presumes that the locations of customers and the traveling time among customers are fixed and constant. In real-life cases, however, the traffic conditions and customer requests may change over time. To find the most economic route, the decisions can be made constantly upon the time-point when the salesman completes his service of each customer. This brings in a dynamic version of the traveling salesman problem (DTSP), which takes into account the information of real-time traffic and customer requests. DTSP can be extended to a dynamic pickup and delivery problem (DPDP). In this article, we ameliorate the attention model to make it possible to perceive environmental changes. A deep reinforcement learning algorithm is proposed to solve DTSP and DPDP instances with a size of up to 40 customers in 100 locations. Experiments show that our method can capture the dynamic changes and produce a highly satisfactory solution within a very short time. Compared with other baseline approaches, more than 5% improvements can be observed in many cases.**

***Index Terms*— Attention model, deep reinforcement learning (DRL), dynamic traveling salesman problem (DTSP), machine learning, policy gradient.**

I. Introduction

A TRAVELING salesman problem (TSP) is a famous problem in the area of computer science, operational research, transportation, and automation. It targets finding the shortest route visiting all the customers in different locations exactly once. In the original TSP, the problem structure remains unchanged, implying that the locations of customers and the traveling time among customers are fixed. However, in most real-world scenarios, we have to deal with route planning in dynamic environments, which make the problem much more challenging.

In literature, there are some relevant studies on dynamic traveling salesman problems (DTSPs), e.g., [1]–[4], among which two typical types attract much research attention. The first one is DTSP with changing customers [5], [6]. This arises in some scenarios such as pick-up services: the requests show up when the customers place orders and disappear if they are canceled. The second one is DTSP with changing traffic [7], [8]. The changing traffic would affect the expected total traveling time of a route and sometimes distance due to a forced detour. It is worth noting that there is a similar problem called a time-dependent traveling salesman problem (TDTSP) [9]. In this problem, the traveling time from customer i to j depends on the moment when the salesman completes the visit of customer i. However, TDTSP is a single-stage decision-making problem, i.e., a route is determined before the start of the travel and the visiting sequence cannot be altered.

In this article, we consider a sequential decision-making problem of DTSP, where the salesman can decide his next visited customer after the visit of the current one is completed. We also take into account both customer and traffic changes. A customer may be dynamically added into or deleted from the unvisited pool after the salesman serves a customer. We assume that the salesman would not respond to the customer changes and make decisions when he is traveling on the way, although we can easily modify our method to allow the decision making at any time. The traffic condition of each edge (i, j) is varied from time to time according to the time-dependent function $g_{ij}(t)$ in addition to a stochastic variable $\phi_{ij}(t)$ on the edge. We further extend DTSP to its variant called a dynamic pick-up and delivery problem (DPDP), in which a customer request is separated into a pair of pick-up and delivery requests. The pick-up request should be made prior to the corresponding delivery request. All the requests must be completed within a route.

To tackle dynamic routing problems, most of the existing approaches are based on meta-heuristics, e.g., [3], [10], [11]. They mainly focus on the dynamic entrance of customers where new customers may appear unpredictably. They first divide the planning horizon into multiple periods. Then a routing decision is made at the beginning of each period. One representative framework for dynamic routing problems is referred to as the ant colony optimization with immigrant schemes [12], [13]. However, it cannot handle the issue of

Manuscript received 26 August 2020; revised 23 March 2021 and 2 June 2021; accepted 11 August 2021. Date of publication 14 September 2021; date of current version 5 April 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2018AAA0101203, in part by the National Science Foundation of China under Grant 71601191 and Grant 62072483, in part by the Guangdong Natural Science Funds under Grant 2019A1515011169 and Grant 2021A1515011301, and in part by the Ministry of Science and Higher Education of the Russian Federation as part of World-class Research Center Program: Advanced Digital Technologies under Contract 075-15-2020-903. *(Corresponding author: Hong Liu.)*

Zizhen Zhang and Jiahai Wang are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China, and also with the Guangdong Key Laboratory of Big Data Analysis and Processing, Sun Yat-sen University, Guangzhou 510275, China (e-mail: zhangzzh7@mail.sysu.edu.cn; wangjiah@mail.sysu.edu.cn).

Hong Liu is with the School of Data Science, City University of Hong Kong, Hong Kong, China (e-mail: honliu4-c@my.cityu.edu.hk).

MengChu Zhou is with Helen and John C. Hartmann Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA, and also with the Department of Cyber-Physical Systems, Saint Petersburg State Marine Technical University, 198262 Saint Petersburg, Russia (e-mail: zhou@njit.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3105905.

Digital Object Identifier 10.1109/TNNLS.2021.3105905

dynamic traffic, and its time complexity is high due to its need for a large number of iterations. In some real-time planning scenarios such as those in controlled airspace [14], the environment could change rapidly. Hence, we are keen to find a real-time approach for tackling DTSPs. To this end, we utilize a deep reinforcement learning (DRL) paradigm, which can solve a wide range of complex online decision-making problems with uncertainty and dynamicity within a much shorter time than meta-heuristics [15]. Although DRL is computationally expensive in training a network model (known as learning offline), once the model is well-trained, the decisions can be made quickly via a forward pass through the network (known as planning online).

Recently, with the rapid development of machine learning, some ingenious and inspiring DRL approaches are proposed to deal with TSPs and other combinatorial optimization problems. Most of these approaches are based on an encoder–decoder framework [16]. While an encoder converts all the inputs to some learned vectors, a decoder transforms the learned vectors to the outputs. Seq2seq [17], as one of the most widely used encoder–decoder frameworks, treats the inputs (e.g., the customer locations in TSP) and outputs (e.g., the best route of TSP) as two sequences. A recurrent neural network (RNN) is used in both encoder and decoder. Since the output of a routing problem depends on the customer positions in the input, Vinyals *et al.* [18] proposed a pointer network framework. It uses a content-based attention mechanism as a pointer to choose an element of the input sequence for output. Both Seq2seq and pointer network approaches utilize RNNs, which may, unfortunately, cause gradient vanishing/explosion problems in some instances with long sequences. The studies [19], [20] proposed attention models to alleviate such issue. Purely using attention mechanisms suffices to produce very promising results compared with meta-heuristic approaches. In recent years, graph neural networks (GNNs) [21], which treat the input as a graph structure, are applied to solve TSP in sparse graphs. Various variants of GNNs are then proposed to solve TSPs and other combinatorial optimization problems, such as a graph attention network [22] and graph convolutional network [23].

To train a DRL model for maximizing the expected long-term reward (being equivalent to minimizing the expected total traveling distance in DTSP), some typical methods, e.g., policy gradient [24] and Q-learning [25], [26] are introduced. Policy gradient is a policy-based approach, where the optimal policy is directly learned from an agent interacting with the environment. Q-learning is a value-based approach that can infer the optimal policy from the learned value function based on Bellman's optimality equation. In addition, deep deterministic policy gradient [27], actor-critic [28], and policy gradient with baselines [20] are also widely used within DRL frameworks.

Next, we briefly introduce the literature related to online routing problems with DRL approaches. Yu *et al.* [29] proposed a DRL mechanism with an unsupervised auxiliary network to solve an online vehicle routing problem. Mao and Shen [30] studied an adaptive routing problem in the stochastic and time-dependent networks. An online Q-learning algorithm and an offline fit Q-iteration algorithm are investigated. Hu *et al.* [31] study a multiple TSP. They devised a shared GNN and distributed policy networks to learn a common policy representation to produce near-optimal solutions to the problem. The work [32] presents a DRL approach for the capacitated arc routing problem. A graph convolutional network and two encoder–decoder models are integrated into the approach. There also exist several studies incorporating the heuristic search within the DRL framework. For example, in Lu *et al.* [33], a DRL approach is proposed to select a sequence of operators to improve the initial solution for vehicle routing problems.

Our work can be summarized as follows. It proposes a DRL approach with two encoder–decoder models M1 and M2 by some modifications of the existing models [18], [19]. Specifically, 1) the static attribute and dynamic one of a state are separated; 2) both M1 and M2 take traffic patterns into consideration; 3) the encoder–decoder architecture of M2 is very different from the existing literature; and 4) the pretraining and fine-tuning of M2 are needed. With these modifications, the proposed approach is able to perceive dynamic environmental changes that are not well-considered in existing work. Experimental results show that it can better deal with dynamic routing problems than existing approaches. It can produce promising solutions in a significantly short time for practical cases under the study. As a result, the proposed framework can be easily adopted in real-time planning applications [15], [34].

The remainder of this article is organized as follows. In Section II, we provide a formal definition of DTSP and DPDP. In Section III, we introduce a DRL approach with two deep neural network (NN) models for solving the problem. To evaluate and compare our approach with several baseline algorithms, Section IV presents the computational results on benchmark data. Section V gives some closing remarks.

II. Problem Definition

DTSP is defined on a complete bidirected graph $G = (V, E)$, where V is a node set with size n and E is an edge set. V consists of a depot 0 and a set of potential customers. We consider the asymmetric distance in DTSP. Thus, E includes edges in either direction. The customers to be visited are put into a customer pool C with size c, where C is a subset of V.

The salesman starts his trip from depot 0 at the beginning of time ($t = 0$). He has to serve each customer in pool C exactly once and then goes back to the depot. The traveling time from node i to node j depends on a time-dependent function $g_{ij}(t)$, where t is the time visiting node i. We assume that the salesman does not wait at a node. This is valid when the First-In–First-Out constraint [35] is respected, i.e., it guarantees that if a vehicle leaves a node i for a node j at a given time, any identical vehicle leaving node i for node j at a later time will arrive later at node j.

Let x_{ij} be a binary decision variable which is equal to 1 if the salesman travels from node i to node j, and 0 otherwise.

Let s_i be the time that the salesman visits node i. The objective is to minimize the total traveling time for completing all the customer visits, that is

$$\min \sum_{i\in\{0\}\cup C} \sum_{j\in\{0\}\cup C} g_{ij}(s_i)x_{ij}. \tag{1}$$

The constraint set is

$$\sum_{j\in\{0\}\cup C} x_{ji} = 1 \quad \forall i \in C \tag{2}$$

$$\sum_{j\in\{0\}\cup C} x_{ij} = 1 \quad \forall i \in C \tag{3}$$

$$s_0 = 0 \tag{4}$$

$$s_i + g_{ij}(s_i)x_{ij} = s_i + (s_j - s_i)x_{ij} \quad \forall i \in \{0\}\cup C, j \in C \tag{5}$$

$$x_{ij} \in \{0, 1\}. \tag{6}$$

Constraints (2) and (3) ensure that there is only one incoming and outgoing node for customer i. Constraint (4) is the starting time of the salesman at the depot. Constraints (5) indicate that the visiting time of customer j depends on the visiting time of its predecessor i. This set of constraints also guarantees that the visiting time at each node is increasing along the path (given that $g_{ij}(t) > 0$). Consequently, there exists no subtour in the solution.

The model expressed by (1)–(6) is essentially a TDTSP formulation. It is nonlinear due to the time-dependent function $g_{ij}(t)$. Some work tries to linearize the formulation by imposing additional assumptions [36], [37]. Different from them, our work studies the most generalized version. Note that the domain of $g_{ij}(t)$ is continuous. For ease of data collection, we discretize the time horizon into a set T of timesteps. In this manner, we have the traveling time from node i to node j around timestep $t \in T$ as the input values, denoted as d_{ijt}. Here, we call $[d_{ijt}]$ the traffic pattern of graph G. Then we can approximate $g_{ij}(t)$ by operating on d_{ijt} (see Section IV-A for more details).

TDTSP presumes that all the traffic conditions are known in advance. In practice, to handle a dynamic environment, we introduce a stochastic variable $\phi_{ij}(t)$ in addition to $g_{ij}(t)$ to address the uncertainty issue of real-time traffic. Then the actual traveling time from node i to node j at time t, denoted as $f_{ij}(t)$, is $f_{ij}(t) = g_{ij}(t) + \phi_{ij}(t)$.

To address another uncertainty issue, i.e., the change of customer requests in a dynamic environment, we involve a random operation $\boldsymbol{\Omega}_k$ after a salesman completes the service of the kth customer, denoted as

$$\boldsymbol{\Omega}_k = \begin{cases} 1, & \text{insert an unvisited customer } i \text{ into set } C \\ 0, & \text{do nothing} \\ -1, & \text{delete a customer } i \text{ from set } C. \end{cases}$$

DTSP is an online optimization problem. Solving it in an efficient way is very difficult. Consider the problem with the scale $n = 40$ of a location invariant graph G. If $c = 20$, the number of possible instances is as large as C_{40}^{20}. When the two kinds of aforementioned dynamic issues are considered, the problem becomes even more challenging.

We next describe a variant of DTSP called DPDP. In this problem, the customer set C is composed of two sets: a pick-up request set C^+ and a delivery request set C^-. For ease of description, we assume that the ith ($1 \le i \le c/2$) customer request is in C^+, while the $(i + c/2)$th ($1 \le i \le c/2$) customer request is the corresponding delivery requests in C^-. For each pick-up request i, q_i specifies the number of goods to be transported. For the corresponding delivery request $i + c/2$, we set $q_{i+c/2} = -q_i$. A salesman (or vehicle) has a capacity Q. We need to discover a route to serve all the customers, ensuring that pick-up requests are visited before their corresponding delivery requests, while the capacity limit is not violated.

Similar to DTSP, the change of traffic and customer requests is also considered in DPDP. In particular, the setting of a random operation $\boldsymbol{\Omega}_k$ after the kth visit is as follows:

$$\boldsymbol{\Omega}_k = \begin{cases} 1, & \text{insert an unvisited pick-up request } i \text{ into set } C^+ \text{ and the associated delivery request } i + c/2 \text{ into set } C^- \\ 0, & \text{do nothing} \\ -1, & \text{delete a pick-up request } i \text{ from set } C^+ \text{ and the associated delivery request } i + c/2 \text{ from set } C^-. \end{cases}$$

To exclude the uncertainty issue, a time-dependent PDP (TDPDP) formulation is given as follows. The objective function (1) and constraint (2)–(6) are kept. In addition, denote l_i as the vehicle's load after visiting customer i. The following constraints are needed:

$$s_i \le s_j \quad \forall i \in C^+, j \in C^-, j = i + c/2 \tag{7}$$

$$l_0 = 0 \tag{8}$$

$$l_i + q_j x_{ij} = l_i + (l_j - l_i)x_{ij} \quad \forall i \in \{0\}\cup C, j \in C \tag{9}$$

$$0 \le l_i \le Q \quad \forall i \in C. \tag{10}$$

Constraints (7) indicate that the pick-up time of a request must be prior to the associated delivery time. Constraints (8) and (9) present the dependence of the vehicle load in the route. Constraints (10) give the range of the vehicle load. Similar to the relation between DTSP and TDTSP, DPDP is an online optimization version of TDPDP.

III. Solution Method

In order to solve DTSPs in a real-time manner, we propose a DRL approach adapted from the attention model [20]. The approach mainly includes a model building part and a model training method. In what follows, we first present the proposed DRL approach. We then detail how we design two encoder–decoder architectures to cope with changing traffic and customer requests. For easy descriptions, Table I summarizes the notations to be used in our solution method.

A. DRL Approach

Deep NNs have a strong capability in feature extraction, while reinforcement learning approaches are good at learning profitable actions. DRL combines the merits of both. In this

TABLE I
SUMMARY OF NOTATIONS

T	the set of all timesteps
n	the size of V (i.e., the total number of nodes)
c	the size of C (i.e., the total number of customers)
N	the number of attention layers
x_i	the identity of node i, an n-dimensional vector using the one-hot encoding
X_i	the embedding of node i, a d_h-dimensional vector ($d_h = 128$)
d_{ijt}	the estimated traveling time from node i to node j at timestep t
$g_{ij}(t)$	the estimated traveling time between nodes i and j at time t
y_i	the estimated traffic condition of edges outgoing from node i, $y_i = [d_{ijt},\ \forall j \in V, \forall t \in T]$, a $n \cdot \lvert T\rvert$-dimensional vector
Y_i	the embedding of traffic condition at node i, an d_h-dimensional vector
$h_i^{(j)}$	the input of the j^{th} layer, also the output of the $(j-1)^{th}$ layer, a d_h-dimensional vector
h_i	the embedding of node i, a d_h-dimensional vector
h	the embedding of whole graph, a d_h-dimensional vector
$[a, b]$	the concatenation of two vectors a, b
l	the last visited node
$w_i^{(t)}$	the estimated traffic condition of node i at time t, i.e., $w_i^{(t)} = [g_{ij}(t), \forall j \in V]$, a n-dimensional vector
$v_i^{(t)}$	the indicator of whether node i is visited before time t (0: unvisited, 1: visited)
$w^{(t)}$	the feature of the current traffic (time t), a d_h-dimensional vector
$v^{(t)}$	the feature of the customer visited status, a d_h-dimensional vector
t_k	the time of the k^{th} visit
$p_i^{(t)}$	the probability of node i to be selected as the next node to visit at time t

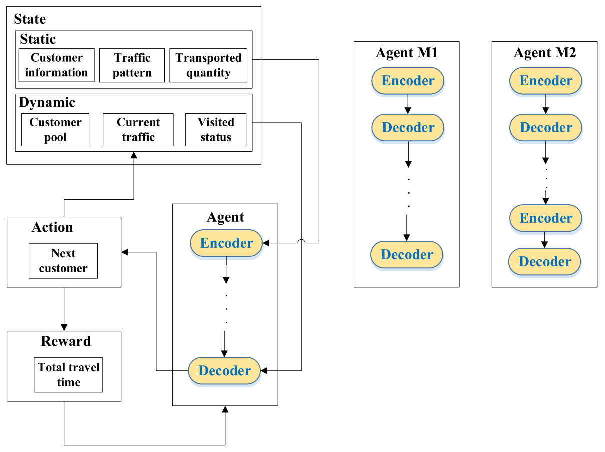


Fig. 1. Illustration of DRL framework.

article, the proposed DRL framework treats DTSPs as a Markov decision process. It consists of four components, i.e., state, agent, action, and reward, as shown in Fig. 1.

The state is formed by the fusion of static and dynamic attributes. The static attributes include customer information (e.g., their identities and coordinates), traffic patterns and the transported quantity of each request. The dynamic attributes contain the current customer pool, visited status of customers and current traffic conditions. The agent observes the state information, and decides which action to take. We propose two alternative encoder–decoder models for the agent. An encoder–decoder model, parameterized by θ, is to train a stochastic policy $p_\theta(\pi|s)$, which denotes the probability of choosing tour π given a state s. According to the probability chain rule, $p_\theta(\pi|s)$ can be computed as

$$p_\theta(\pi|s) = \prod_{k=1}^{c} p_\theta(\pi_k|s, \pi_1, \ldots, \pi_{k-1}).$$

The first model (M1) is similar to the one in [20]. It has one encoder and c decoders sharing the same network. The encoder is fed with the static information of initial c customers and then generates an embedding to represent the graph context. The decoders sequentially output each π_k based on the context embedding, current traffic information, and visited status of customers. M1 mainly targets sensing dynamic traffic, although it can be extended to handle dynamic customer requests.

The second model (M2) has c_M encoders and c_M decoders, where c_M is the maximum number of customers in the dynamic customer pool. All these encoders (resp., decoders) share an identical network. In each step, M2 re-encodes the state information and outputs the next customer to visit. Because entering or exiting customers can be captured by the encoder, the experiments show that M2 performs better than M1 on dynamic request scenarios. However, M2 is difficult to train due to its deep architecture. Therefore, several training tricks have to been adopted. In summary, the comparison of M1 and M2 is presented in Table II.

Once an agent has decided on the next customer to visit, the state dynamics are updated and the corresponding reward can be collected. Since the objective is to minimize the total traveling time, the immediate reward is set to the opposite of the traveling time of the traversed edge.

We utilize a standard policy gradient with rollout baseline algorithm [20] for model training, as realized in Algorithm 1. It involves two network models for a particular agent: θ represents the current model and θ^* represents the best model found so far. These two models have the same structure. A one-sided pair t-test is employed to compare the performance of these two models on a batch of (e.g., 10 000) randomly generated test instances. If θ is significantly better than θ^*, then θ^* is updated. Otherwise, if θ^* is not updated for M epochs, θ is rolled back to θ^*. In the algorithm, model θ^* is served as a baseline, which can assist to reduce the variance of results for different instances.

B. Model 1 (M1)

The architecture of M1 is shown in Fig. 2. We elaborate the details of its encoder and decoder as follows.

TABLE II
COMPARISON BETWEEN MODEL M1 AND M2

	M1	M2
Number of encoding steps	1	c_M
Number of decoding steps	c	c_M
Re-encoding the state after each visit	No	Yes
Pretraining the submodels	None	Node2Vec, Traffic2Vec
Advantage	Fast training, better handling dynamic traffic	Better handling dynamic customer requests

Algorithm 1 Proposed Policy Gradient With Rollout Baseline Algorithm

Parameter: No. epochs E, No. samples per epoch S, rollback epoch M
Input: Objective function z
Output: Trained model θ^*

```
1:  Initialize the model $\theta$, $\theta^* \leftarrow \theta$
2:  ep$\leftarrow$ 0
3:  for epoch = 1, ..., $E$ do
4:    $x_i \leftarrow$ RandomInstance(), $\forall i \in 1, \ldots, S$
5:    $\pi_i \leftarrow \theta(x_i)$ $\forall i \in 1, \ldots, S$
6:    $\pi_i^* \leftarrow \theta^*(x_i)$ $\forall i \in 1, \ldots, S$
7:    $\nabla$loss $\leftarrow \sum_{i=1}^{S}(z(\pi_i) - z(\pi_i^*))\nabla_\theta \log p_\theta(\pi_i)$
8:    $\theta \leftarrow$ Adam($\theta, \nabla loss$)
9:    if OneSidedPairedTTest($z_\theta, z_\theta^*$) <0.05 then
10:     $\theta^* \leftarrow \theta$, ep$\leftarrow$epoch
11:   else
12:     if epoch $-$ ep $> M$ then
13:       $\theta \leftarrow \theta^*$
14:     end if
15:   end if
16: end for
17: return $\theta^*$
```

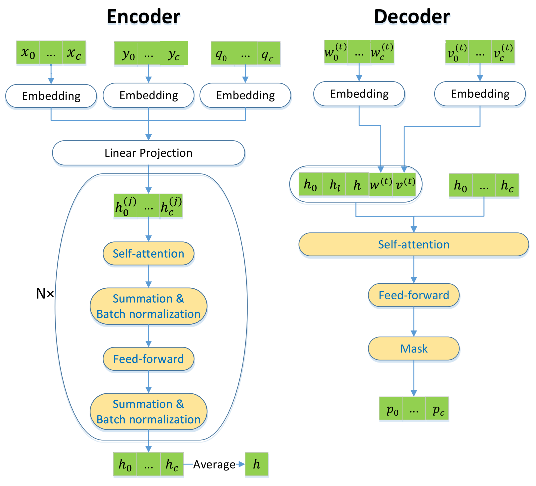


Fig. 2. Proposed model M1.

1) Encoder: The encoder takes the identity of a node and estimated traffic pattern (i.e., the traveling time between each pair of nodes along the time horizon) as the input, and outputs the embedding (or features) of each node as well as the whole graph. Specifically, it first converts the identity of each customer node i into ID embedding X_i and converts the estimated traffic condition into traffic embedding Y_i through a fully connected NN. For DPDP, the transported quantity q_i is also embedded. Next, it concatenates X_i and Y_i and produces the hidden embedding of each node, i.e., $h_0^{(0)}, \ldots, h_c^{(0)}$. Finally, the encoder computes the aggregated embedding h_i by N attention layers, each of which contains a multihead attention (MHA) sublayer and a node-wise fully connected feed-forward (FF) sublayer with one hidden layer and ReLu activation. The following formulations describe the proposed encoder. The details of MHA sublayer, which are the same as [20], are provided in Appendix A.

a) Formulations:

$$\begin{aligned}
X_i &= \mathcal{N}(x_i) \quad \forall i \in \{0\} \cup C\\
Y_i &= \mathcal{N}(y_i) \quad \forall i \in \{0\} \cup C\\
h_i^{(0)} &= \mathcal{N}([X_i, Y_i]) \quad \forall i \in \{0\} \cup C\\
\hat{h}_i^{(j)} &= \mathcal{B}\Big(h_i^{(j-1)} + \mathcal{A}\Big(h_0^{(j-1)}, \ldots, h_c^{(j-1)}\Big)\Big) \quad \forall i \in \{0\} \cup C\\
h_i^{(j)} &= \mathcal{B}\Big(\hat{h}_i^{(j)} + \mathcal{F}\Big(\hat{h}_i^{(j)}\Big)\Big) \quad \forall i \in \{0\} \cup C\\
h_i &= h_i^{(N)} \quad \forall i \in \{0\} \cup C
\end{aligned}$$

where $\mathcal{N}$ represents a fully connected NN layer, $\mathcal{F}$ represents an FF sublayer with one hidden layer and ReLu is an activation fuction, $\mathcal{B}$ represents a batch normalization sublayer, and $\mathcal{A}$ represents an MHA sublayer.

2) Decoder: Like traditional RNNs [16], the proposed decoder is reused in c steps. For each step, it outputs the next visited customer. In particular, it takes the current traffic conditions, visited status of customers, and the graph embeddings computed by the encoder as the input, and outputs the probabilities of all customers to be selected as the next visited customer.

Specifically, it first produces the embedding $w^{(t)}$ from the current traffic information with respect to each node i (i.e., the traveling time from node i to other nodes at time t), and the embedding $v^{(t)}$ from the node masks (i.e., whether a node is visited or not). Then, five embeddings $w^{(t)}$, $v^{(t)}$, h, h_0, and the feature of last visited node h_l are concatenated to represent the context of the state at time t. Next, the attention mechanism (using one attention layer) is applied to the context feature and other node features. Finally, it masks all the visited nodes and calculates the probability of the next unvisited node with a softmax layer. It is worth noting that for DPDP, the following two cases further influence the mask mechanism:

1) If the transported quantity of a pick-up request exceeds the remaining capacity of the vehicle, this request should be masked.
2) If a pick-up request has not started yet, its associated delivery request should also be masked.

After the next visited node is determined, the next timestep t can be realized and the node mask is updated accordingly. Thereafter, the next decoding step can be applied until all the customers are successfully visited. The below formulations can well show the decoder in this article.

a) Formulations:

$$
\begin{aligned}
w^{(t)} &= \mathcal{N}\left(w_0^{(t)}, \ldots, w_c^{(t)}\right)\\
v^{(t)} &= \mathcal{N}\left(v_0^{(t)}, \ldots, v_c^{(t)}\right)\\
g &= \left[w^{(t)}, v^{(t)}, h, h_0, h_l\right]\\
\left[h'_0, \ldots, h'_c\right] &= \mathcal{F}(\mathcal{A}(g, h_0, \ldots, h_c))\\
\left[p_0^{(t)}, \ldots, p_c^{(t)}\right] &= \operatorname{softmax}\left(\left[h'_0, \ldots, h'_c\right] \otimes \left[v_0^{(t)}, \ldots, v_c^{(t)}\right]\right)
\end{aligned}
$$

where "$\otimes$" represents the element-wise multiplication.

3) Extension to Dynamic Requests: Once the model is trained, we can do parameter sharing to cope with dynamic requests in the model testing. When a new customer request, say x_k, arises, we first obtain its embedding h_k using the trained network. Then the attention mechanism in the decoder can further take h_k into consideration. When a customer request is removed from the pool, we can simply change its corresponding visited status to 0 (i.e., visited). In this case, the mask mechanism can automatically filter out this request. Because M1 does not learn how the customer changes from the training data, it appears that M1 is not very competitive in the dynamic customer request environment.

C. Model 2 (M2)

The major drawback of M1 lies in its context embedding h used in its decoder. Because h is obtained from its encoder, it cannot foresee dynamic requests tackling in the decoder part. We, therefore, propose M2, which re-encodes the states after each visit to capture the change of customer requests.

Both encoder and decoder of M2 are reused in c_M steps alternatively. Fig. 3 illustrates the architecture of a pair of them. Training M2 can be divided into two stages: pretraining and fine-tuning. Such an idea is inspired by a popular framework BERT [38] in natural language processing. The reason why we introduce the pretraining strategy is that it is very difficult to train a model into the one that converges and produces satisfactory solutions when both customer and traffic changing information is input into the encoder.

1) Pretraining Stage: In this stage, two models Node2Vec and Traffic2Vec are, respectively, trained using Algorithm 1. Their targets are to obtain the embeddings of node information and traffic information, respectively. By comparing the architectures of M2, Node2Vec, and Traffic2Vec, there are only minor differences among their encoders, while the structures of their decoders are the same. For a decoder, it takes the context embeddings (the embeddings of graph h, depot h_0 and last visited node h_l), and node embeddings h_i as the input, and

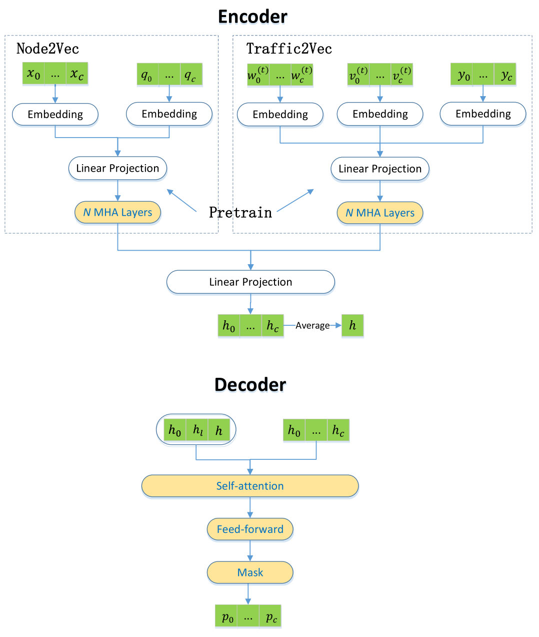


Fig. 3. Proposed model M2.

output the probability p_i of the next node to visit. Different from M1, the embeddings of the current traffic and customer visited status ($w^{(t)}$ and $v^{(t)}$) are removed from the decoders, as they are stacked in the encoder part.

For Node2Vec, we pretrain a model by using all the n nodes instead of c nodes. This can be done by properly using a mask mechanism: those nodes not shown in an instance are masked. Doing so can help us obtain the embeddings of those dynamic customer requests. For easy training, the traveling time between each pair of nodes f_{ij} is assumed to be independent of the traffic condition and calculated as

$$f_{ij} = \mathbb{E}_{t\sim T}\left[f_{ij}(t)\right] \approx \frac{\sum_{t\in T} d_{ijt}}{|T|}.$$

For Traffic2Vec, we pretrain a model by using c nodes as the same in M1, inputting the traffic conditions at time t ($w_i^{(t)}$), the visited status of node i at time t ($v_i^{(t)}$), and the traffic pattern (y_i). After each decoder, we re-encode the state with the updated traffic conditions.

Note that Traffic2Vec only considers the information about traffic, while the information about nodes is ignored. Training Traffic2Vec is difficult enough. Therefore, it is not wise to input both node information and traffic conditions into the network, as it would become too complicated to train it to achieve the desired convergence. This is the reason why we split our framework into two separate stages.

2) Fine-Tuning Stage: In this stage, the two pretrained models Node2Vec and Traffic2Vec are fixed. They assist to obtain the embeddings of the node and traffic in DTSPs. These embeddings are then fused to produce the graph embedding h with two fully connected NNs, as shown in Fig. 3. Except for the pretrained Node2Vec and Traffic2Vec, the decoder and the remaining part of the encoder in M2 are trained in this stage.

3) Coping With Dynamic Requests: An intriguing property of the proposed model is that it can be induced from static environments, and applied to dynamic environments. The reason is that static environments suffice to cover most aspects of the dynamic environments using the state information (the customer pool, traffic conditions, and visited status).

In the model testing, we have c customers in the pool initially to feed into the encoder. For the unused $c_M - c$ nodes in the encoder, we set their visited status v_i to 1 (i.e., unvisited). After a customer is visited, it happens to delete or insert a customer. For deletion, we only need to set the corresponding visited status to 0 (i.e., visited). For insertion, we exchange the embedding of the inserted node with the embedding of an unused node. The traffic conditions are also updated for the next encoder–decoder pair.

IV. Experiments

Our DRL approaches are programed with Pytorch. They are executed on a workstation with Intel(R) Xeon(R) CPU E5-2680 v4 clocked at 2.40 GHz and a single GTX1080Ti GPU.

A. Benchmark Dataset

To evaluate the performance of our approach, we propose a benchmark dataset for DTSP, which is extracted from a real-world dataset in Beijing provided by Jingdong Logistics. We randomly select 1 depot and 99 customers from the dataset. Fig. 4 shows their locations on the map, where the depot is located at the bottom-right of the map. The mean of the traveling time between a pair of nodes is 51.16 min. Using these 100 nodes, we post requests to Baidu Map API every two hours of a day to get the time-dependent traveling time matrix (or traffic pattern) $[d_{ijt}]_{100\times100\times12}$.

To generate synthetic real-time traffic information $f_{ij}(t)$ on each edge (i, j), we adopt the cubic spline interpolation (CSI) [39]. Specifically, we use 12 discrete points $(0, d_{i,j,1})$, $(2, d_{i,j,2})$, $\ldots$, $(22, d_{i,j,12})$ to obtain 11 cubic functions $S_k(t) = S_{k,0} + S_{k,1}t + S_{k,2}t^2 + S_{k,3}t^3, (k = 0, 1, \ldots, 11, 2k \le t \le 2(k+1))$. These functions are combined to obtain the estimated traveling time function $g_{ij}(t)$. To simulate the uncertainty, we add a stochastic value $\phi_{ij}(t)$ to represent unpredictable incidents. Denote $\phi_{\max}, \phi_{\min}$ and σ as three hyperparameters, and the real-time traffic function $f_{ij}(t)$ is given as

$$f_{ij}(t) = g_{ij}(t) + \max\big(\text{LB}, \min\big(\text{UB}, \phi_{ij}(t)\big)\big) \tag{11}$$

$$\text{LB} = \phi_{\min} \cdot g_{ij}(t) \tag{12}$$

$$\text{UB} = \phi_{\max} \cdot g_{ij}(t) \tag{13}$$

$$\phi_{ij}(t) \sim N\big(0, \sigma^2\big). \tag{14}$$

Finally, we generate the data of changing customer requests. After the kth visit, we set Ω_k to -1 with ρ^- probability, 1 with ρ^+ probability, and 0 with $1-\rho^- - \rho^+$ probability. If Ω_k equals -1, we randomly delete a customer from the unvisited customer pool C. If Ω_k equals 1, we randomly add an unvisited customer into C.

We also generate a DPDP dataset using the same way. Additional parameters are set as follows. The vehicle capacity Q is set to 20 or ∞ (unlimited). The quantity q_i of each pick-up customer i is a random integer between 1 and 9.

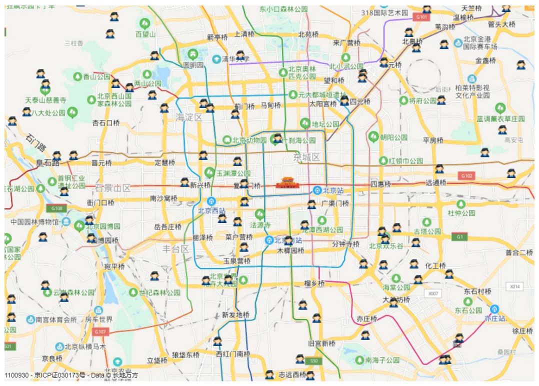


Fig. 4. Location map of the benchmark dataset.

TABLE III
Computational Time of Different Approaches

c	SA	DP	GR	M1	M2	Solver
4	0.02s	0.01s	8.53e-4s	5.33e-5s	**8.61e-5s**	0.01s
9	0.14s	0.23s	8.72e-4s	7.94e-5s	**1.35e-4s**	0.13s
19	1.03s	3.59s	9.01e-4s	1.31e-4s	**4.27e-4s**	0.53s

B. Baseline Algorithms and Models

To our best knowledge, there is no existing method that can solve DTSPs directly. We propose four baseline algorithms and compare them with our DRL approach.

1) Offline simulated annealing (SA): It consists of five neighborhood operators: 2-opt, exchange, relocate, or-opt, and 3-opt. These operators are widely used in routing problems [40]. To evaluate the performance of SA, we first ran it on a real-world TDTSP dataset in Lyon [35]. The preliminary results indicate that the solution produced by SA is at most 2% worse than the optimal solution. In the following experiments, we only apply SA in DTSP with changing traffic. More details of SA are provided in Appendix B.
2) Greedy (GR): It is an online greedy algorithm, which selects the nearest and unvisited node as the next node to visit.
3) Online TSP Solver (Solver): It is an online TSP algorithm adapted to DTSP. After each visit, the algorithm finds the optimal subroute considering the current traveling time matrix by using a TSP solver. Here, we apply Concorde[2] to find the optimal route for TSP instances.
4) Online Dynamic Programming (DP): It is an online algorithm adapted from Held-Karp algorithm [41]. After each visit, DP finds the optimal route based on the time-dependent function $g_{ij}(t)$, but the real traveling time is realized according to $f_{ij}(t)$. Note that DP is an exact method for solving TDTSP or TDPDP. However, because of the existence of uncertainty, DP appears to be a heuristic approach. The time complexity of DP is $O(c^2 2^c)$ for a single decision. Therefore, it is impossible to directly apply it to the scenarios with more than

[2]https://github.com/jvkersch/pyconcorde/

TABLE IV
COMPUTATIONAL RESULTS WITH RESPECT TO DIFFERENT σ AND c ON THE FIRST DTSP SCENARIO

ϕ_{min}	ϕ_{max}	σ	c	SA	Solver	DP	GR	M1	M2	M1-FF	M2-NP
-0.9	5	43.2	4	283.23	287.33	286.14	298.92	**235.80**	241.80	395.42	298.38
-0.9	5	43.2	9	430.67	435.31	433.10	459.04	**388.85**	396.40	714.85	439.54
-0.9	5	43.2	19	728.03	724.43	726.63	788.38	**682.05**	685.92	831.15	771.77
Avg.	-	-	-	480.64	482.36	481.96	515.45	**435.57**	441.37	647.14	503.23
-0.5	0.5	43.2	19	573.11	566.10	569.22	636.45	**546.42**	548.85	700.40	658.60
-0.5	2	43.2	19	735.88	726.58	731.98	793.90	**700.67**	704.16	853.94	808.49
-0.5	5	43.2	19	791.22	787.42	789.97	849.18	**756.16**	758.25	900.76	850.98
-0.5	10	43.2	19	794.73	798.58	800.69	859.24	**766.99**	768.94	908.35	857.87
-0.9	0.9	43.2	19	575.51	568.59	573.75	639.44	**536.15**	542.20	694.82	642.34
-0.9	2	43.2	19	670.18	663.40	668.00	733.17	**626.36**	631.87	783.20	727.86
-0.9	5	43.2	19	728.03	724.43	726.63	788.38	**682.05**	685.92	831.15	771.77
-0.9	10	43.2	19	736.20	735.71	737.88	798.81	**693.24**	696.55	838.71	778.98
-1	1	43.2	19	573.30	569.01	574.74	640.52	**533.40**	540.16	693.17	638.26
-1	2	43.2	19	653.15	650.79	655.24	721.00	**611.06**	616.95	768.97	711.74
-1	5	43.2	19	716.00	711.88	714.47	776.36	**666.85**	671.12	817.20	755.95
-1	10	43.2	19	724.37	723.19	725.72	786.89	**677.91**	681.56	824.67	762.81
Avg.	-	-	-	689.30	685.47	689.02	751.95	**649.77**	653.88	801.28	747.14
-0.9	5	0	19	564.24	561.88	561.47	632.80	560.82	**560.30**	706.55	678.18
-0.9	5	14.4	19	587.44	580.57	582.79	650.96	**551.15**	551.85	703.33	654.68
-0.9	5	28.8	19	649.49	643.81	647.20	711.88	**607.28**	610.39	757.53	703.88
-0.9	5	43.2	19	728.03	724.43	726.63	788.38	**682.05**	685.92	831.15	771.77
-0.9	5	N(14.4, 36^2)	19	674.80	670.88	670.55	735.19	**630.23**	635.92	788.37	725.59
-0.9	5	N(14.4, 72^2)	19	813.51	786.00	789.12	842.33	**742.81**	750.20	904.56	837.98
-0.9	5	N(28.8, 36^2)	19	710.68	704.50	708.41	771.93	**664.77**	668.38	821.31	765.10
-0.9	5	N(28.8, 72^2)	19	815.05	808.42	812.12	861.80	**757.36**	762.82	920.94	850.49
-0.9	5	N(43.2, 36^2)	19	755.56	756.97	762.06	820.15	**711.55**	715.62	867.73	808.76
-0.9	5	N(43.2, 72^2)	19	835.77	832.20	837.42	890.83	**784.30**	786.88	946.91	879.89
-0.9	5	U(0, 28.8)	19	589.81	587.38	591.02	659.32	**559.89**	561.93	711.91	664.08
-0.9	5	U(0, 57.6)	19	661.36	656.48	658.03	724.83	**620.64**	626.59	773.18	718.24
-0.9	5	U(0, 86.4)	19	732.37	730.47	735.47	795.71	**691.34**	697.43	843.93	784.67
Avg.	-	-	-	701.39	695.69	698.64	760.47	**658.78**	662.63	813.65	757.18

Notes: N(μ, ϵ^2) is a normal distribution with minimum value 0 and maximum value 2μ, and U(a, b) is a uniform distribution in $[a, b]$.

30 customers. Please refer to Appendix C for more details of DP.

Note that SA is served as a single-stage decision-making approach. Thus it is essentially solving TDTSP, but not suitable to tackle the case of changing customer requests. Although the other three algorithms are multistage decision-making approaches and can be applied to DTSPs, GR and Solver may be vulnerable in the environment with traffic uncertainty issues, and DP would be too slow to produce a solution in a timely manner.

In addition, to prove the efficacy of the attention model and the pretraining strategy, we modify them to derive two other baseline learning models for comparison.

1) M1-FF: It is modified from model M1 by substituting the attention layers with a fully connected FF layer.
2) M2-NP: Instead of pretraining two submodels Node2Vec and Traffic2Vec, it trains the whole model M2 directly.

C. Computational Results

We apply Algorithm 1 to the model training of M1, Node2Vec, Traffic2Vec, and M2, respectively. The related parameters are set as follows: $E = 1000$, $S = 1\,00\,000$ and $M = 100$. Then, we test our models on different scenarios. For each scenario, there are 10 000 cases for the model testing.

We consider the first scenario: DTSP with changing traffic only. We first test the average computational time of different approaches on the 100-node DTSP instances with $c = 4, 9$ and 19, respectively. Here, we set $\Omega_k \equiv 0$ in (7), and set $\phi_{\min} = -0.9$, $\phi_{\max} = 5$, $\sigma = 43.2$ in (11)–(14). The results of computational time are shown in Table III. As we can see, our approaches (M1 and M2) can obtain a solution in a millisecond level. DP is not able to handle the instance with $c = 39$ or even larger.

The solution quality (i.e., the total traveling time on average) is provided in Table IV. The upper part of Table IV corresponds to the instances used in Table III. The remaining part corresponds to the instances with $c = 19$. This is because $c = 19$ is a proper size to evaluate the performance of different approaches. We also test different settings of $\phi_{\min}$, $\phi_{\max}$ and σ. In particular, a constant σ indicates that all the edges follow the same distribution. Otherwise, different edges may be associated with different distributions.

TABLE V

AVERAGE RANK OF THE ALGORITHMS ON THE FIRST DTSP SCENARIO

Algorithm	Average Ranking Value	Final Rank
M1	1.04	1
M2	1.96	2
Solver	3.31	3
DP	4.27	4
SA	4.42	5
M2-NP	6.27	6
GR	6.73	7
M1-FF	8	8

From Table IV, we find that M1 and M2 outperform other candidate algorithms in all the cases. The results clearly verify the robustness of M1 and M2 in this scenario. The performances of SA, Solver, and DP are good enough, while GR, M1-FF, and M2-NP are not quite competitive against other approaches. M1 produces very promising solutions (about 5.3% better than Solver and 5.7% better than DP on average). M2 performs slightly (about 1.0%) worse than M1 since no dynamic change of customers is considered in this scenario. To statistically test whether the proposed models are significantly better than other approaches, we conduct the Friedman test [42]. In the test, we calculate the average rank of each algorithm, as shown in Table V. The Friedman's chi-square value $\tau_f = 579.60$, which means that the performance of the algorithm is significantly different from others. Therefore, we next conduct the Nemenyi test [42]. The critical difference value $CD = 2.059$, which means that if the rank difference between two algorithms is larger than CD, the performance of the two algorithms significantly differs. According to the test, M1 is significantly better than all the competitors except M2.

In order to intuitively illustrate why M1 can perceive the dynamic traffic environment and avoid traffic congestions, we may visualize two example solutions whose details are presented in Appendix D.

Next, we test our method on the second scenario: DTSP with both customer and traffic changes. We set different σ, c_0, ρ^+ and ρ^- for the instances with $c_M = 19$ (the maximum number of customers in the pool). In this set of experiments, if the number of customers in the pool exceeds c_M, we would ignore the corresponding operation (i.e., $\Omega_k = 1$).

The results are provided in Table VI. From this table, we can find that the performance of M2 is close to that of DP. The more dynamic insertion of customer requests (i.e., the cases with higher ρ^+), the smaller difference between them. Compared with M1, M2 is more robust to customer changes, as it performs significantly better than M1 for most of the cases. In some cases where no new customer appears ($\rho^+ = 0$), M1 performs better than M2. The reason is that M1 does not recalculate the embedding after each customer. On average, M2 can produce a solution that is 0.9% better than DP and 1.4% better than GR. As a matter of fact, M2 is an efficient model to solve DTSP with dynamic customer requests. In this scenario, we also conduct the

TABLE VI

COMPUTATIONAL RESULTS WITH RESPECT TO DIFFERENT σ, c_0 AND ρ ON THE SECOND DTSP SCENARIO

σ	c_0	ρ^+	ρ^-	Solver	DP	GR	M1	M2
0	17	5	5	544.99	541.19	588.75	569.87	**538.62**
0	17	10	10	550.73	545.19	578.00	589.28	**535.72**
0	15	10	0	541.36	537.83	590.35	599.75	**537.22**
0	19	0	10	532.87	**530.54**	575.24	532.35	531.49
14.4	17	5	5	558.60	557.00	601.67	578.88	**546.78**
14.4	17	10	10	566.05	561.23	592.37	597.17	**546.32**
14.4	15	10	0	554.75	552.89	602.21	608.65	**547.46**
14.4	19	0	10	547.10	546.94	588.14	**541.62**	542.51
28.8	17	5	5	611.67	609.93	652.11	627.23	**595.05**
28.8	17	10	10	617.80	612.17	642.72	642.86	**595.06**
28.8	15	10	0	605.25	603.60	650.36	651.32	**595.83**
28.8	19	0	10	601.86	601.44	642.04	**594.68**	597.19
43.2	17	5	5	681.36	679.04	721.23	694.81	**662.07**
43.2	17	10	10	685.57	682.52	710.08	707.06	**662.84**
43.2	15	10	0	672.57	669.50	716.43	712.91	**662.17**
43.2	19	0	10	675.13	674.40	712.95	**665.41**	669.60
Avg.	-	-	-	593.68	590.83	635.29	619.62	**585.37**

TABLE VII

AVERAGE RANK OF THE ALGORITHMS ON THE SECOND DTSP SCENARIO

Algorithm	Average Ranking Value	Final Rank
M2	1.25	1
DP	2.125	2
Solver	3.25	3
M1	3.75	4
GR	4.625	5

Friedman–Nemenyi test. The results are shown in Table VII. In the test, $\tau_f = 36.61$ and $CD = 1.52$. From the table, we notice that M2 is significantly better than Solver, M1, and GR, since their rank differences are greater than CD. Besides, the computational time of M2 is very short. Therefore, we can safely conclude that M2 is the best solver in this scenario.

Lastly, we evaluate our algorithms on DPDP with both traffic and customer changes. Different combinations of σ, Q, c_0, c_M, ρ^+ and ρ^- are tested. The results are provided in Table VIII. For DPDP instances, M2 still provides the best policy. For the instances with $c = 20$, M2 is 0.9% better than DP and significantly better than M1 and GR. For the instances with $c = 40$, M2 is 13.9% better than GR and 13.6% better than M1.

D. Analysis of Different Training Configurations

We first show some training details of models M1 and M2. Figs. 5 and 6 display the L2-norm of gradient (grad_norm) and expected reward (i.e., total traveling time) in the training period for the case with $\sigma = 0$ and $c = 19$. As we can see, the gradient decreases steadily as the number of epoches increases. The spike in the grad_norm plot reflects that the model parameters change rapidly in that epoch. With respect to the expected reward of M1, it significantly drops from 640th to 560th epoches. Correspondingly, the expected reward

TABLE VIII
COMPUTATIONAL RESULTS ON DPDP INSTANCES

σ	Q	c_0	c_M	ρ^+	ρ^-	DP	GR	M1	M2
0	∞	18	20	10	10	**603.84**	662.05	664.95	610.21
0	20	18	20	10	10	**625.00**	688.31	696.77	615.20
14.4	∞	18	20	10	10	619.12	673.10	669.45	**617.78**
14.4	20	18	20	10	10	640.77	702.59	692.57	**622.17**
28.8	∞	18	20	10	10	671.69	714.77	711.27	**665.39**
28.8	20	18	20	10	10	687.94	743.35	729.30	**667.86**
43.2	∞	18	20	10	10	736.16	783.93	773.28	**730.98**
43.2	20	18	20	10	10	752.53	810.93	787.24	**733.96**
0	∞	20	20	0	10	**583.75**	649.00	611.21	600.71
0	20	20	20	0	10	**605.64**	672.11	643.45	609.31
14.4	∞	20	20	0	10	**601.43**	659.64	616.73	607.60
14.4	20	20	20	0	10	623.70	687.29	646.84	**614.96**
28.8	∞	20	20	0	10	656.18	704.44	663.40	**655.95**
28.8	20	20	20	0	10	677.37	730.49	687.44	**662.59**
43.2	∞	20	20	0	10	727.53	780.39	730.46	**723.64**
43.2	20	20	20	0	10	740.03	796.78	748.34	**729.00**
0	∞	16	20	10	0	**645.35**	744.57	746.70	670.45
0	20	16	20	10	0	**670.71**	766.14	770.28	670.72
14.4	∞	16	20	10	0	**661.48**	715.33	742.14	665.73
14.4	20	16	20	10	0	686.73	777.26	764.93	**669.66**
28.8	∞	16	20	10	0	716.11	792.23	780.87	**707.80**
28.8	20	16	20	10	0	735.15	821.05	799.69	**715.13**
43.2	∞	16	20	10	0	791.37	861.37	843.03	**773.02**
43.2	20	16	20	10	0	812.23	887.38	858.31	**783.01**
Avg.	-	-	-	-	-	677.99	742.69	724.10	**671.78**
0	20	36	40	10	10	-	1133.72	1151.68	**958.78**
14.4	20	36	40	10	10	-	1153.30	1171.06	**982.97**
28.8	20	36	40	10	10	-	1230.71	1251.14	**1087.61**
43.2	20	36	40	10	10	-	1357.27	1373.44	**1214.92**
0	20	40	40	0	10	-	1132.46	1065.06	**942.55**
14.4	20	40	40	0	10	-	1150.85	1083.42	**971.86**
28.8	20	40	40	0	10	-	1233.51	1167.36	**1072.68**
43.2	20	40	40	0	10	-	1355.88	1292.39	**1202.67**
0	20	32	40	10	0	-	1279.95	1318.54	**1068.25**
14.4	20	32	40	10	0	-	1298.48	1332.48	**1091.54**
28.8	20	32	40	10	0	-	1387.00	1426.10	**1195.53**
43.2	20	32	40	10	0	-	1530.78	1568.09	**1340.74**
Avg.	-	-	-	-	-	-	1270.33	1266.73	**1094.18**

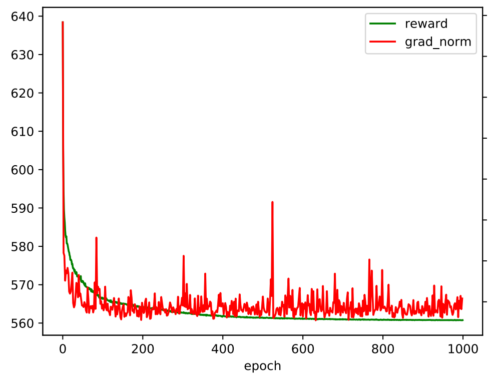


Fig. 5. Plot of gradient norm and reward in training M1.

of M2 drops from 590th to 560th epoches. The figure verifies the convergence of the model training. We can also observe that the convergence plots of M1 and M2 are both smooth, but in the first epoch, M2 decreases much more than M1. This observation indicates that the pretraining stage is helpful. The submodels (i.e., Node2Vec and Traffic2Vec) can learn the latent features of customer requests and traffic patterns.

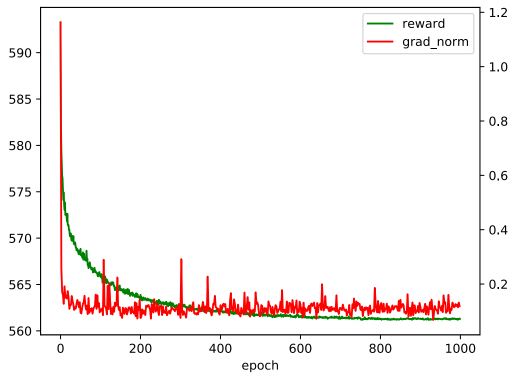


Fig. 6. Plot of gradient norm and reward in training M2.

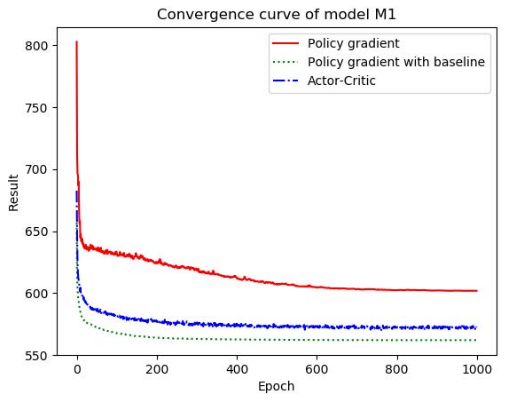


Fig. 7. Results of different training algorithms.

Different training algorithms are also tested, including policy gradient, policy gradient with baseline, and actor–critic. Fig. 7 shows their convergence curves. From it, we find that all the training algorithms can make the model converge steadily. The proposed policy gradient with baseline performs slightly better than actor-critic, and much better than policy gradient. This verifies the effectiveness of our approach.

V. Conclusion

In this article, we study a DTSP and its related pickup and delivery problem, which consider the change of traffic conditions and customer requests in dynamic environments. In order to solve the problem in a real-time manner, we treat them as sequential decision-making problems. A reinforcement learning approach, which incorporates two deep NN models, is proposed. Benchmark datasets extracted from a real-world application are introduced to evaluate the efficiency of the approach. Experiments verify that our approach can much better to tackle TSP and PDP in dynamic environments and with uncertainty issues than other existing methods.

Our solution method can be easily modified to deal with different real-world dynamic routing problems, e.g., dynamic versions of other vehicle routing problems [2], [43]–[46].

TABLE IX

PERFORMANCE GAP BETWEEN SA AND CP ON TDTSP INSTANCES

n	Scenario 1	Scenario 2	Scenario 3
10	0.35%	0.29%	0.51%
20	0.75%	0.70%	1.15%
30	0.10%	0.16%	0.69%
50	1.12%	0.84%	1.82%

Algorithm 2 SA Algorithm

Input: objective function $z(\pi)$

Parameter: initial temperature $T_{\text{init}} = 100$, initial solution π_{init}, cooling rate $\alpha = 0.7$

Output: a near-optimal route

```
1:  $T \leftarrow T_{\text{init}}, \pi^* \leftarrow \pi_{\text{init}}$
2:  while the stopping criterion not reached do
3:    repeat
4:      $\pi_{last} \leftarrow \pi^*$
5:      for $i \leftarrow 1$ to 5 do
6:        $\pi \leftarrow \pi^*$
7:        for $j \leftarrow 1$ to $n^2$ do
8:          randomly choose a neighbor solution $\pi'$ generated by operator $O_i$ on $\pi$
9:          if $z(\pi') < z(\pi)$ or rand() $< e^{\frac{z(\pi)-z(\pi')}{T}}$ then
10:           $\pi \leftarrow \pi'$
11:         end if
12:         if $z(\pi') < z(\pi^*)$ then
13:           $\pi^* \leftarrow \pi'$
14:         end if
15:       end for
16:     end for
17:   until $\pi^* = \pi_{\text{last}}$
18:   $T \leftarrow \alpha T$
19: end while
20: return $\pi^*$
```

While most of the previous approaches are iteration-based and difficult to apply in various dynamic environments, our method is learning-based and flexible. We only need to revise our encoder–decoder architecture to make it perceivable to other features of the environments. In addition, spatio-temporal GNNs [47] could be introduced to make more accurate traffic predictions in dynamic environments. However, how to design a tailored deep NN that has excellent generalization capabilities in dealing with different scale problems, and is easy to train via various approaches [48]–[52] remains open.

APPENDIX

A. MHA Layer

The MHA layer propagates the information between nodes. With $M = 8$ heads, the nodes can receive messages from different perspectives.

1) Notations:

1) d_h: The number of hidden dimensions (e.g., 128)
2) d_k: The number of key dimensions, $d_k = (d_h/M)$
3) h_i: The ith input embedding, a d_h-dimensional vector
4) m: The index of attention head
5) q_{im}: The mth query vector of the ith input, a d_k-dimensional vector
6) k_{im}: The mth key vector of the ith input, a d_k-dimensional vector
7) v_{im}: The mth value vector of the ith input, a d_k-dimensional vector
8) W_m^Q: The mth query matrix, a $d_h * d_k$ matrix
9) W_m^K: The mth key matrix, a $d_h * d_k$ matrix
10) W_m^V: The mth value matrix, a $d_h * d_k$ matrix
11) u_{ijm}: A temporary d_k-dimensional vector
12) a_{ijm}: The mth attention between node i and j, a d_k-dimensional vector

Algorithm 3 Online DP Algorithm

Input: actual travel time functions $f_{ij}(t)$, estimated travel time functions $g_{ij}(t)$

Output: a near-optimal route

```
1: $t \leftarrow 0, \pi \leftarrow \{0\}$, next $\leftarrow 0$, last $\leftarrow 0$
2: for $i \leftarrow 1$ to $c$ do
3:   $C_{\min}$, next $\leftarrow$Held-Karp($g$, last, $t$, $\pi$, None)
4:   $\pi \leftarrow \pi \cup \{\text{next}\}, t \leftarrow t + f_{\text{last,next}}(t)$
5:   last $\leftarrow$ next
6: end for
7: return $\pi$
```

Algorithm 4 Held-Karp Algorithm

Input: estimated travel time function $g_{ij}(t)$, last visited node i, the current time t, the set of visited customers π, an array DP representing the best route starting from customer i with visited customer set π.

Output: the minimal cost and the next node to visit

function Held-Karp()

```
1:  if DP is None then
2:    initialize DP with value $-1$
3:  end if
4:  if DP[$i$][$\pi$] $\geq 0$ then
5:    return DP[$i$][$\pi$], None
6:  end if
7:  if $\pi$ contains all customers then
8:    DP[$i$][$\pi$] $\leftarrow g_{i0}(t)$
9:    return $g_{i0}(t)$, 0
10: end if
11: $C_{\min} \leftarrow \infty$, next $\leftarrow -1$
12: for $j \leftarrow 1$ to $c$ do
13:   if $j$ is not in $\pi$ then
14:     $p, k$ = Held-Karp($g$, $j$, $t + g_{ij}(t)$, $\pi \cup \{j\}$, DP)
15:     if $p + t < C_{\min}$ then
16:       next $\leftarrow j$
17:       $C_{\min} \leftarrow p + t$
18:     end if
19:   end if
20: end for
21: DP[$i$][$\pi$] $= C_{\min}$
22: return $C_{\min}$, next
```

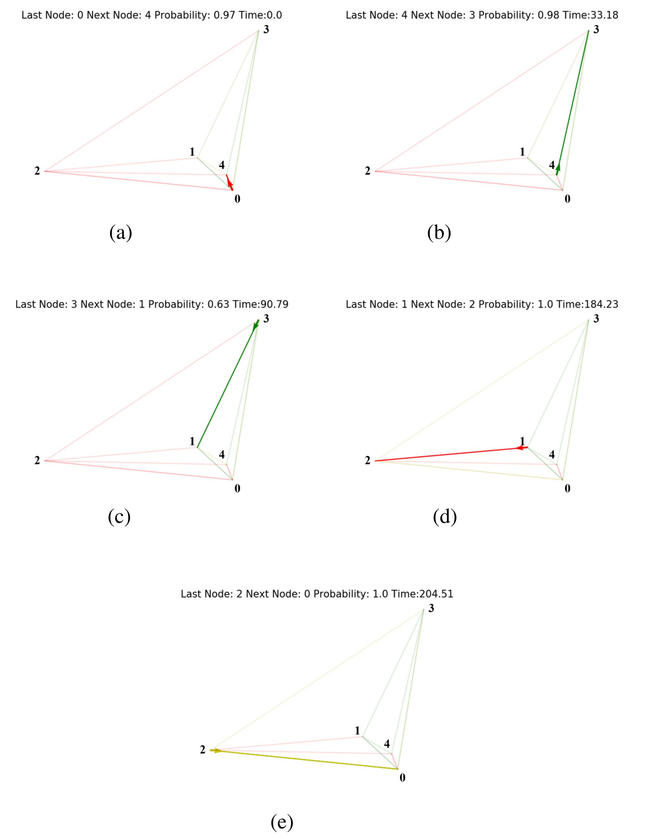

Last Node: 0 Next Node: 4 Probability: 0.97 Time:0.0
Last Node: 4 Next Node: 3 Probability: 0.98 Time:33.18
Last Node: 3 Next Node: 1 Probability: 0.63 Time:90.79
Last Node: 1 Next Node: 2 Probability: 1.0 Time:184.23
Last Node: 2 Next Node: 0 Probability: 1.0 Time:204.51
(a)
(b)
(c)
(d)
(e)

Fig. 8. First example solution. (a) First visit. (b) Second visit. (c) Third visit. (d) Fourth visit. (e) Fifth visit.

13) W_m^O: The mth output matrix, a $d_k * d_h$ matrix
14) MHA_i: The ith output embedding, a d_h-dimensional vector

2) Formulations:

$$\begin{aligned} q_{im} &= W_m^Q h_i \\ k_{im} &= W_m^K h_i \\ v_{im} &= W_m^V h_i \\ u_{ijm} &= q_{im}^T k_{jm} \\ a_{ijm} &= \frac{e^{u_{ijm}}}{\sum_{k=0}^{n-1} e^{u_{ikm}}} \\ \text{MHA}_i(h_0, \ldots, h_{n-1}) &= \sum_{m=1}^{M} W_m^O \sum_{j=0}^{n-1} a_{ijm} v_{jm}. \end{aligned}$$

B. SA Algorithm

The pseudo code of the proposed SA is presented in Algorithm 2. Five neighborhood operators are used: O_1: 2-opt, O_2: exchange, O_3: relocate, O_4: or-opt, O_5: 3-opt. The performance gap between SA and an exact constraint programming (CP) on the TDTSP dataset [35] is provided in Table IX. The results indicate that the solution obtained by SA is at most 2% worse than the optimal solution.

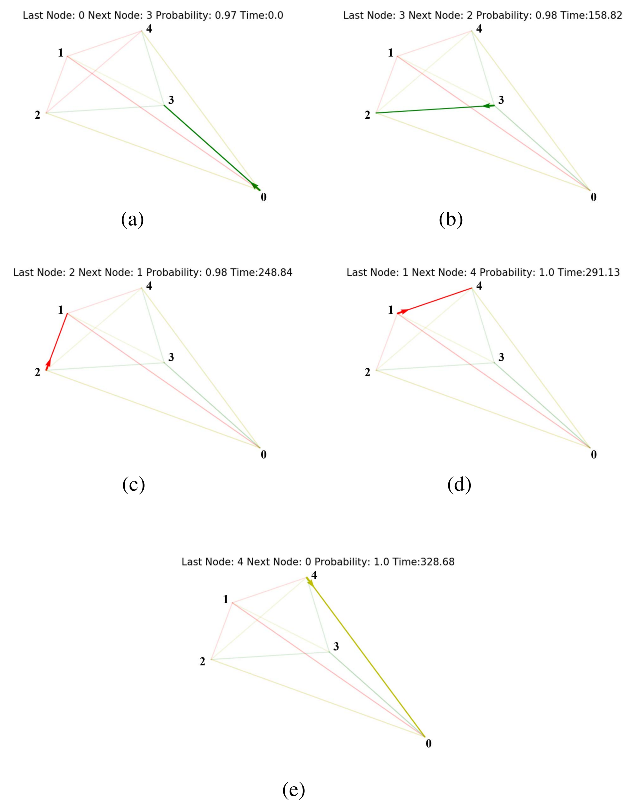

Last Node: 0 Next Node: 3 Probability: 0.97 Time:0.0
Last Node: 3 Next Node: 2 Probability: 0.98 Time:158.82
Last Node: 2 Next Node: 1 Probability: 0.98 Time:248.84
Last Node: 1 Next Node: 4 Probability: 1.0 Time:291.13
Last Node: 4 Next Node: 0 Probability: 1.0 Time:328.68
(a)
(b)
(c)
(d)
(e)

Fig. 9. Second example solution. (a) First visit. (b) Second visit. (c) Third visit. (d) Fourth visit. (e) Fifth visit.

C. Dynamic Programming

The online DP algorithm for TDTSP is presented in Algorithm 3. After each visit, it invokes a subroutine Held-Karp algorithm [41], as shown in Algorithm 4, to decide the optimal route based on the time-dependent function $g_{ij}(t)$. Note that the real traveling time is depending on $f_{ij}(t)$.

D. Example Solution Visualization

In Figs. 8 and 9, we provide two example solutions on two 5-node DTSP instances obtained by the model M1. In these figures, the red, yellow and green lines represent that the corresponding edges are in heavy, normal, and light traffic, respectively.

As we can see, for most of the time, the DRL agent would choose the most promising edge (light traffic) with high confidence to travel. In Fig. 8(c), it is difficult for the DRL agent to decide whether the next visited customer is 1 or 2, since the expected traveling time between the partial route (3,1,2,0) and (3,2,1,0) is very close.

REFERENCES

[1] L. Abbatecola, M. P. Fanti, and W. Ukovich, "A review of new approaches for dynamic vehicle routing problem," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2016, pp. 361–366.
[2] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *Eur. J. Oper. Res.*, vol. 225, no. 1, pp. 1–11, 2013.

[3] M. Mavrovouniotis and S. Yang, "Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem," in *Evolutionary Computation for Dynamic Optimization Problems*. Berlin, Germany: Springer, 2013, pp. 317–341.

[4] S. Jiang and S. Yang, "A benchmark generator for dynamic multi-objective optimization problems," in *Proc. 14th UK Workshop Comput. Intell. (UKCI)*, Sep. 2014, pp. 1–8.

[5] M. Guntsch and M. Middendorf, "Applying population based ACO to dynamic optimization problems," in *Proc. Int. Workshop Ant Algorithms*. Berlin, Germany: Springer, 2002, pp. 111–122.

[6] R. Necula, M. Breaban, and M. Raschip, "Tackling dynamic vehicle routing problem with time windows by means of ant colony system," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2017, pp. 2480–2487.

[7] C. J. Eyckelhof and M. Snoek, "Ant systems for a dynamic TSP," in *Proc. Int. Workshop Ant Algorithms*. Berlin, Germany: Springer, 2002, pp. 88–99.

[8] T. Cheong and C. C. White, "Dynamic traveling salesman problem: Value of real-time traffic information," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 2, pp. 619–630, Jun. 2012.

[9] C. Malandraki and M. S. Daskin, "Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms," *Transp. Sci.*, vol. 26, no. 3, pp. 185–200, 1992.

[10] C. Groba, A. Sartal, and X. H. Vázquez, "Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices," *Comput. Oper. Res.*, vol. 56, pp. 22–32, Apr. 2015.

[11] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, "Ant colony system for a dynamic vehicle routing problem," *J. Combinat. Optim.*, vol. 10, no. 4, pp. 327–343, Dec. 2005.

[12] S. Gao, Y. Wang, J. Cheng, Y. Inazumi, and Z. Tang, "Ant colony optimization with clustering for solving the dynamic location routing problem," *Appl. Math. Comput.*, vol. 285, pp. 149–173, Jul. 2016.

[13] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors," *Appl. Soft Comput.*, vol. 13, no. 10, pp. 4023–4037, 2013.

[14] F. Furini, C. A. Persiani, and P. Toth, "The time dependent traveling salesman planning problem in controlled airspace," *Transp. Res. B, Methodol.*, vol. 90, pp. 38–55, Aug. 2016.

[15] T. Wu *et al.*, "Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 8243–8256, Aug. 2020.

[16] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[18] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2692–2700.

[19] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[20] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[21] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2702–2711.

[22] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[23] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 539–548.

[24] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[25] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] L. Jiang, H. Huang, and Z. Ding, "Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 4, pp. 1179–1189, Jul. 2020.

[27] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6348–6358.

[28] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9839–9849.

[29] J. J. Q. Yu, W. Yu, and J. Gu, "Online vehicle routing with neural combinatorial optimization and deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 10, Oct. 2019, pp. 3806–3817.

[30] C. Mao and Z. Shen, "A reinforcement learning framework for the adaptive routing problem in stochastic time-dependent network," *Transp. Res. C, Emerg. Technol.*, vol. 93, pp. 179–197, Aug. 2018.

[31] Y. Hu, Y. Yao, and W. S. Lee, "A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs," *Knowl.-Based Syst.*, vol. 204, Sep. 2020, Art. no. 106244. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705120304445

[32] H. Li and G. Li, "Learning to solve capacitated arc routing problems by policy gradient," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 1291–1298.

[33] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[34] Z. Ning *et al.*, "Joint computing and caching in 5G-envisioned internet of vehicles: A deep reinforcement learning-based traffic control system," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5201–5212, Aug. 2021.

[35] P. A. Melgarejo, P. Laborie, and C. Solnon, "A time-dependent no-overlap constraint: Application to urban delivery problems," in *Proc. Int. Conf. AI OR Techn. Constriant Program. Combinat. Optim. Problems*. Cham, Switzerland: Springer, 2015, pp. 1–17.

[36] J.-F. Cordeau, G. Ghiani, and E. Guerriero, "Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem," *Transp. Sci.*, vol. 48, no. 1, pp. 46–58, Feb. 2014.

[37] A. Montero, I. Méndez-Díaz, and J. J. Miranda-Bront, "An integer programming approach for the time-dependent traveling salesman problem with time Windows," *Comput. Oper. Res.*, vol. 88, pp. 280–289, Dec. 2017.

[38] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1. Stroudsburg, PA, USA: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[39] S. A. Dyer and J. S. Dyer, "Cubic-spline interpolation. 1," *IEEE Instrum. Meas. Mag.*, vol. 4, no. 1, pp. 44–46, Mar. 2001.

[40] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, Part I: Route construction and local search algorithms," *Transp. Sci.*, vol. 39, no. 1, pp. 104–118, Feb. 2005.

[41] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *J. ACM*, vol. 9, no. 1, pp. 61–63, 1962.

[42] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.

[43] J. Li, M. C. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2390–2401, Nov. 2015.

[44] L. Wang and J. Lu, "A memetic algorithm with competition for the capacitated green vehicle routing problem," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 2, pp. 516–526, Feb. 2019.

[45] J. Wang, Y. Sun, Z. Zhang, and S. Gao, "Solving multitrip pickup and delivery problem with time windows and manpower planning using multiobjective algorithms," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 4, pp. 1134–1153, Jul. 2020.

[46] X. Xu, J. Li, and M. Zhou, "Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 3, pp. 1583–1593, Mar. 2021.

[47] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 1–7.

[48] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019.

[49] L. Shao, D. Wu, and X. Li, "Learning deep and wide: A spectral method for learning deep networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 12, pp. 2303–2308, Dec. 2014.

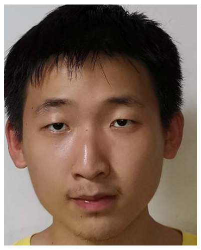
[50] X. Luo, X. Wen, M. Zhou, A. Abusorrah, and L. Huang, "Decision-tree-initialized dendritic neuron model for fast and accurate data classification," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 17, 2021, doi: 10.1109/TNNLS.2021.3055991.

[51] T. Liu, B. Tian, Y. Ai, and F. Wang, "Parallel reinforcement learning-based energy efficiency improvement for a cyber-physical system," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 2, pp. 617–626, Mar. 2020.

[52] G. Wang, J. Qiao, J. Bi, W. Li, and M. Zhou, "TL-GDBN: Growing deep belief network with transfer learning," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 874–885, Apr. 2019.

Zizhen Zhang received the B.S. and M.S. degrees from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2009, respectively, and the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2014.

He is currently an Associate Professor with the School of Computer Science and Engineering, Sun Yat-sen University. His research interests include computational intelligence, reinforcement learning, and various applications in production, transportation and logistics.



Hong Liu received the M.S. degree from the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China, in 2020. He is currently pursuing the Ph.D. degree with the School of Data Science, City University of Hong Kong, Hong Kong.

His research interests include but not limited to reinforcement learning, combinatorial optimization, and wind forecasting.

MengChu Zhou (Fellow, IEEE) received the B.S. degree from Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree from Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined New Jersey Institute of Technology, Newark, NJ, USA, in 1990, where he is currently a Distinguished Professor in electrical and computer engineering. He has over 900 publications, including 12 books, 600+ journal articles (500+ in IEEE TRANSACTIONS), 29 patents, and 29 book-chapters. His recently co-authored/edited books, include *Supervisory Control and Scheduling of Resource Allocation Systems: Reachability Graph Perspective* (Hoboken, NJ, USA: IEEE Press/Wiley, 2020) (with B. Huang) and *Contemporary Issues in Systems Science and Engineering* (Hoboken, NJ, USA: IEEE/Wiley, 2015) (with H.-X. Li and M. Weijnen). His research interests include Petri nets, intelligent automation, the Internet of Things, big data, web services, and intelligent transportation.

Prof. Zhou is a fellow of the International Federation of Automatic Control (IFAC), the American Association for the Advancement of Science (AAAS), the Chinese Association of Automation (CAA), and the National Academy of Inventors (NAI). He is a life member of the Chinese Association for Science and Technology, USA, and served as its President in 1999. He was a recipient of Humboldt Research Award for U.S. Senior Scientists from Alexander von Humboldt Foundation, the Franklin V. Taylor Memorial Award, the Norbert Wiener Award from the IEEE Systems, Man and Cybernetics Society, the Excellence in Research Prize and Medal from NJIT, and the Edison Patent Award from the Research and Development Council of New Jersey. He is the Founding Editor of IEEE PRESS BOOK SERIES ON SYSTEMS SCIENCE AND ENGINEERING, the Editor-in-Chief of IEEE/CAA JOURNAL OF AUTOMATICA SINICA, and an Associate Editor of IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, and IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS.

Jiahai Wang (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Toyama, Toyama, Japan, in 2005.

In 2005, he joined Sun Yat-sen University, Guangzhou, China, where he is currently a full Professor with the School of Computer Science and Engineering. He is currently leading an Intelligent Optimization and Learning Laboratory, Sun Yat-sen University. He has published a series of articles in leading conferences and top journals including AAAI, ACL, and IEEE transactions. His main research interests include computational intelligence (deep neural networks and metaheuristics) and its applications. He is a Distinguished Member of CCF, China.