Solving Time-Dependent Traveling Salesman Problem with Time Windows with Deep Reinforcement Learning

Guojin Wu[1], Zizhen Zhang[1,*], Hong Liu[2] and Jiahai Wang[1]

Abstract— **Traveling Salesman Problem (TSP) is a well-known NP-hard combinatorial optimization problem. Recently, many researchers have used deep reinforcement learning to solve it. However, traffic factors are rarely considered in their works, in which the traveling time between customer locations is assumed to be constant over the planning horizon. For many practical scenarios, the traffic conditions between customer locations may change over time due to the impact of traffic patterns. Thus, this paper considers a Time-Dependent Traveling Salesman Problem with Time Windows (TDTSPTW), where the time dependency is obtained by fitting the collected traffic data into real-time traffic function with the interpolation method. We propose a deep reinforcement learning framework to solve TDTSPTW. Extensive experiments on TDTSPTW instances indicate that the proposed method can capture the real-time traffic changes and yield high-quality solutions within a very short time, compared with other typical baseline algorithms.**

I. Introduction

Traveling Salesman Problem (TSP) is an NP-hard [1] problem in combinatorial optimization, which have been extensively studied in the scope of operations research and computer science. In this problem, a salesman starts from the origin location, visits each location exactly once and returns to the origin location. Given the traveling cost matrix between each pair of locations, the goal is to find a shortest route for the salesman.

TSP has many applications such as order delivery service and travel planning. However, TSP does not consider the real-time traffic information in these applications. Because of the impact of traffic patterns, the traveling time between each two locations may change over time. In addition, by taking into account the delivery time window for each customer location, which requires the salesman to visit within the specific time interval, would make the problem much more realistic in practical applications.

In this paper, we consider a Time-Dependent Traveling Salesman Problem with Time Windows (TDTSPTW). To incorporate the time dependency (or real-time traffic changes) factors into the problem, we consider a continuous function $\tau_{ij}(t)$, which represents the traveling time from location i to j when the salesman departs from location i at time t. In addition, there is a time window $[E_i, L_i]$ associated with each customer i. The salesman is not allowed to reach location i after L_i. If the salesman arrives at location i before E_i, he has to wait until it is open.

[1]Guojin Wu, Zizhen Zhang and Jiahai Wang are with School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. (* *corresponding author, email: zhangzzh7@mail.sysu.edu.cn*)

[2]Hong Liu with School of Data Science, City University of Hong Kong, Hong Kong S.A.R, China.

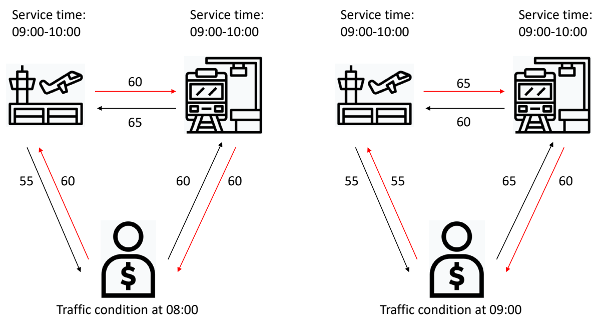
Service time: 09:00-10:00
Service time: 09:00-10:00
60
65
55
60
60
60
Traffic condition at 08:00
Service time: 09:00-10:00
Service time: 09:00-10:00
65
60
55
55
65
60
Traffic condition at 09:00

Fig. 1. An example of TDTSPTW.

For better explanation, we provide an example to illustrate TDTSPTW. Considering the scenario illustrated in Fig. 1, the salesman is required to serve two customers located at the airport and the station at 8:00. If he only takes into account the current traffic condition, he is supposed to follow the red route. This is because if the black route is selected, he thought he would be late for the airport based on the current situation. However, in real-life cases, the traffic condition changes over time. The right part of the figure shows the traffic condition at 9:00. If the salesman chooses the red route as planned, he would be late for the station when he arrives at the airport. Therefore, it would be better for him to choose the black route as no violation occurs in this case.

We treat TDTSPTW as a sequential decision problem, which means that the salesman can iteratively select the unvisited location to travel until all the locations have been visited. To solve the problem in a real-time manner, we propose a deep reinforcement learning framework adapted from [2], [3] to capture the real-time traffic changes. Reinforcement learning enables the agent to learn in a way, trail and error, by interacting with the environment to obtain the maximum accumulated rewards. Once the trained model is converged, it is able to generalize to other similar environments. To test the performance of our framework, we propose two benchmark datasets and baseline algorithms. An ablation study is also provided to verify the significance of each part of the model.

The paper is structured as follows. In Section II, we briefly review the literature of TDTSPTW. Section III defines the problem and introduces a mathematical formulation of TDTSPTW. In Section IV, we propose a deep reinforcement learning approach in details. In order to evaluate its performance, in Section V, we present and discuss the numerical results on the benchmark datasets. In Section VI, we give some closing remarks with suggestions to extend our work.

II. Related Work

In literature, there were some relevant works on time-dependent traveling salesman problems (TDTSP) [4], [5]. TDTSPTW extends TDTSP by specifying that each customer should be visited within a prescribed time window [6].

Many (heuristic) exact algorithms were proposed to obtain (near-) optimal solutions for TDTSPTW. For instance, typical heuristics [7], [8] introduced hand-crafted rules, which often start from one or a group of initial solutions, improve the solution iteratively, and then effectively find a satisfactory solution in a short time. Nevertheless, they are heavily rely on expert knowledge, and thus difficult to guarantee the quality of solutions and generalize to other similar problems. Exact algorithms [9], [10] explore the whole solution space, and find the best solution. However, these techniques are time-consuming. Even with only 40 customers, they take several hours or even days to find (near-) optimal solutions. This fatal shortage makes them difficult to apply in real-time environments. To this end, the trained Deep Learning models may be desirable for solving complex problems quickly.

The interesting idea of applying neural network for combinatorial optimization problems was attributed to Hopfield and Tank [11] in 1985. However, it only recently becomes a hot research area since deep reinforcement learning has achieved great success in games [12]. Vinyals, Fortunato and Jaitly [13], inspired by the encoder-decoder architecture [14] in machine translation, introduced a Pointer Network to solve TSP in the manner of supervised learning. The encoder and decoder both use recurrent neural network (RNN). The encoder inputs the coordinate of cities and converts them to some hidden states. The decoder uses an attention mechanism as a pointer to choose the next (unvisited) city according to the probability distribution and eventually outputs a permutation of the input. Bello et al. [15] modelled TSP as a Markov Decision Process (MDP), utilizing an actor-critic architecture to efficiently train the Pointer Network. They used negative tour length as the reward to improve the future behavior of agent. Nazari et al. [16] believed that it makes no sense to take into account the order of input sequence, so they omit the RNN encoder and directly use the simple embedding as an input instead of the RNN hidden states. Inspired by the success of Transformer Architecture [2] in machine translation, Kool et al. [3] utilized self-attention mechanism to better aggregate potential relationships between nodes, significantly outperforming all the existing learning-based approaches. Besides, some graph neural network (GNN) based approaches were proposed to solve TSPs and related combinatorial optimization problems, including graph attention network [17], graph convolutional network [18] and graph pointer network [19]. To our best knowledge, there is no existing learning-based approaches for TDTSP or TDTSPTW.

III. Problem Definition

TDTSPTW can be defined as follows. Let $G = (V, A)$ denote a complete graph, where the vertex set $V = \{0, 1, 2, \cdots, n\}$ consists of the origin location (node 0) and a set of customer locations. The edge set A is asymmetric due to the bi-directed traffic condition. A salesman departs from node 0, visits a set of unvisited customers exactly once, and finally returns to node 0.

Suppose that the traveling time between locations depends on the time when the travel commences. We denote a continuous function $\tau_{ij}(t)$ on time horizon T as the traveling time from location i to j when the salesman departs from location i at time t. It is worth noting that $\tau_{ij}(t)$ satisfies the FIFO property [7], i.e., $t + \tau_{ij}(t) \leq t' + \tau_{ij}(t')$ for $\forall t \leq t'$. In this case, when the salesman finishes the visit of a customer, he will not wait and go to the next customer immediately. Each customer i has a corresponding time window $[E_i, L_i]$, i.e., the release time E_i and deadline time L_i.

We denote that x_{ij} is a binary indicator on whether the salesman travels from location i to j, t_i is the time at which the salesman visits location i, r_i is equal to 1 if the salesman violates the time window for customer i, 0 otherwise. r is equal to 1 if the solution is infeasible, 0 otherwise. TDTSPTW can be formulated as follows.

$$\min \quad C_1 r + C_2 \sum_{i \in V \setminus \{0\}} r_i + \sum_{i \in V} \sum_{j \in V} [\tau_{ij}(t_i) + WT_i] x_{ij} \tag{1}$$

subject to

$$\sum_{j \in V} x_{ji} = 1, \sum_{j \in V} x_{ij} = 1, \ \forall i \in V \setminus \{0\} \tag{2}$$

$$WT_i = max(E_i - \tau_{ij}(t_i), 0), \ \forall i \in V \setminus \{0\} \tag{3}$$

$$E_i - \mathcal{M} \cdot r_i \leq t_i \leq L_i + \mathcal{M} \cdot r_i, \ \forall i \in V \tag{4}$$

$$\sum_{i \in V \setminus \{0\}} r_i \leq \mathcal{M} \cdot r \tag{5}$$

$$t_0 = 0, 0 \leq t_i \leq T, \forall i \in V \tag{6}$$

$$x_{ij}, r_i, r \in \{0, 1\}, \forall i, j \in V \tag{7}$$

Here, $\mathcal{M}$ is a sufficiently large number. In objective function (1), we impose a penalty $C_1 = 10,000$ if no feasible solution is found. We also impose a penalty $C_2 = 100$ for each violation of time window constraints. The last term is the total time spent of the route. Constraints (2) ensure that the salesman visits each customer exactly once. Constraints (3) indicate that if the salesman arrives early, he needs to wait until E_i. Constraints (4) require that every customer must be visited within the corresponding time interval, if the time window is not violated. Constraint (5) means that the feasibility of the solution depends on the violation of each time window. Constraints (6) – (7) give the range of variables.

IV. Solution Method

We formulate TDTSPTW as a Markov decision process and use deep reinforcement learning (DRL) to train an agent to make a decision at each location.

Fig. 2 shows the overall framework of DRL. Before a decision is made, the system informs the agent the current state, which is composed of the dynamic information and static information. While the dynamic information includes the current traffic conditions and visited status of customers,

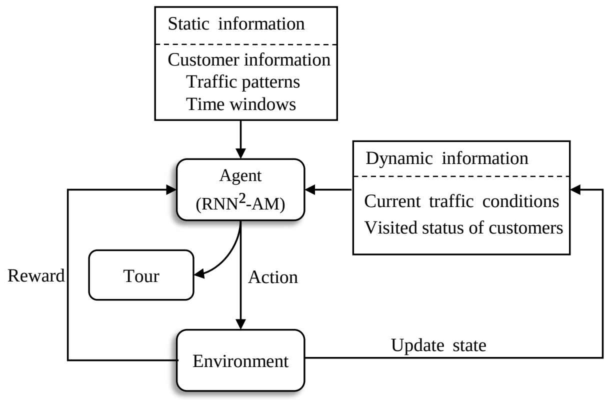

Fig. 2. The diagram of DRL framework.

the static information contains customer information, time windows and traffic patterns. The agent observes the system state, and decides which action (customer location) to choose. After the agent interacts with the environment, the dynamic information will be updated. The total reward is expressed in objective function (1) as an incentive signal for the agent to learn a good strategy.

A. DRL Approach

The proposed DRL approach uses a classic policy-gradient with baseline method to train a deep neural network model, which follows the general encoder-decoder architecture. The model, parameterized by θ, learns a stochastic policy $p_\theta(\pi|s)$ to generate route π given an initial state s. Similar to [3], we use the probability chain rule to compute $p_\theta(\pi|s)$:

$$p_\theta(\pi|s) = \prod_{k=1}^{n} p_\theta(\pi_k|s, \pi_1, ..., \pi_{k-1}) \tag{8}$$

The above equation indicates that the probability of the next selected location determined by θ is conditioned on the previous visited locations.

The overall training process is shown in Algorithm 1. Denote p_{θ^*} as a baseline policy to evaluate the quality of the selected action. During the training process, if the improvement is considered to be significant according to a one-sided pair T-test, the policy p_{θ^*} will be updated with policy p_θ. If θ^* is not updated for M epochs, the current model is rolled back to θ^*.

B. RNN^2-AM Model

The proposed deep neural network model, called RNN^2-AM, is adapted from the classic attention model (AM) [3], in which two RNN modules (an LSTM and a GRU cell) are further incorporated. Because TDTSPTW considers the real-time traffic and has special features along the time horizon. LSTM (or GRU) can help to capture the temporal information of states. The overall architecture of RNN^2-AM is depicted in Fig. 3, where the left part represents the encoder and the right part refers to the decoder. We describe the detailed architecture of the model as follows.

Algorithm 1 The policy gradient with baseline for the model training.

```
1:  Input: objective function $f$, number of epochs $E$, samples per epoch $S$, roll-back epoch $E_R$
2:  Init $\theta$, $\theta^* \leftarrow \theta$, ep$\leftarrow$ 0
3:  for epoch = 1, ... , $E$ do
4:      $x \leftarrow$ RandomInstance()
5:      $\pi_i \leftarrow p_\theta(x_i),\ \forall i \in \{1, ..., S\}$
6:      $\pi_i^* \leftarrow p_{\theta^*}(x_i),\ \forall i \in \{1, ..., S\}$
7:      $\nabla loss \leftarrow \sum_{i=1}^{S}(f(\pi_i) - f(\pi_i^*))\nabla_\theta \log p_\theta(\pi_i)$
8:      $\theta \leftarrow Adam(\theta, \nabla loss)$
9:      if OneSidedPairedTTest($p_\theta, p_{\theta^*}$) <0.05 then
10:         $\theta^* \leftarrow \theta$, ep$\leftarrow$epoch
11:     else
12:         if epoch - ep > $E_R$ then
13:             $\theta \leftarrow \theta^*$
14:         end if
15:     end if
16: end for
17: return $\theta^*$
```

1) Encoder: The encoder consists of three fully connected linear layers, an LSTM layer and N attention layers ($N = 3$) to produce high dimensional representations of customer locations, the estimated traffic pattern (i.e., the traveling time between each pair of locations along the time horizon) and time windows. Firstly, for each location i, the fully connected linear layers map the location identity vector x_i and corresponding time window $[E_i, L_i]$ to ID embedding X_i and time windows embedding TW_i, respectively. Note that the traffic pattern has strong temporal correlation, so the estimated traffic condition y_i (y_i is a sampled vector from the time-dependent function $\tau_{ij}(t), \forall j \in V, t \in T$) is converted into traffic embedding Y_i using an LSTM cell. Then, the whole feature vector $[X_i; Y_i; TW_i]$ is also converted to the hidden embedding of each location h_i^0 using linear projection. The N attention layers further aggregate the correlation between locations to generate the final location embedding h_i^N. Each attention layer consists of a multi-head attention sublayer (MHA) and a fully connected feed-forward sublayer (FF), and each sublayer uses the batch normalization (BN) and residual connection for accelerating deep network training. The output of each attention sublayer h_i^ℓ can be computed as follows:

$$\hat{h}_i^\ell = \mathrm{BN}^\ell(h_i^{\ell-1} + \mathrm{MHA}_i^\ell(h_0^{\ell-1}, \ldots, h_n^{\ell-1})) \tag{9}$$

$$h_i^\ell = \mathrm{BN}^\ell(\hat{h}_i^\ell + \mathrm{FF}^\ell(\hat{h}_i^\ell)) \tag{10}$$

where the MHA component is used to aggregate different types of information from other locations. For each $i \in V$, $\mathrm{MHA}_i^\ell(h_0^{\ell-1}, \ldots, h_n^{\ell-1})$ can be calculated as follows:

$$q_{im}^\ell = W_m^Q h_i^{\ell-1}, k_{im}^\ell = W_m^K h_i^{\ell-1}, v_{im}^\ell = W_m^V h_i^{\ell-1} \tag{11}$$

$$u_{ijm}^\ell = (q_{im}^\ell)^T k_{jm}^\ell \tag{12}$$

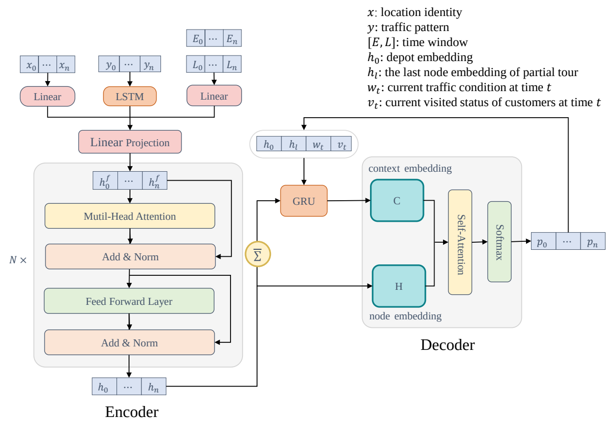

Fig. 3. The proposed model RNN^2-AM.

$$a^{\ell}_{ijm} = \frac{e^{u^{\ell}_{ijm}}}{\sum_{k=0}^{n-1} e^{u^{\ell}_{ikm}}} \tag{13}$$

$$\mathrm{MHA}^{\ell}_i(h^{\ell-1}_0, ..., h^{\ell-1}_n) = \sum_{m=1}^{M} W^O_m \sum_{j=0}^{n-1} a^{\ell}_{ijm} v^{\ell}_{jm} \tag{14}$$

where $M = 8$ is the number of heads in MHA, $W^Q_m \in \mathbb{R}^{d_q \times d_h}, W^K_m \in \mathbb{R}^{d_k \times d_h}$ and $W^V_m \in \mathbb{R}^{d_v \times d_h}$ ($d_q = d_k = d_v = \frac{d_h}{M}$) are trainable matrices. The feed forward sublayer (FF) consists of a fully-connected linear layer with ReLu activation function and another fully-connected linear layer. The process is described as follows:

$$\mathrm{FF}^{\ell}(\hat{h}^{\ell}_i) = W^f_1 \,\mathrm{ReLu}(W^f_0\, \hat{h}^{\ell}_i + b^f_0) + b^f_1 \tag{15}$$

where $W^f_0 \in \mathbb{R}^{d_f \times d_h}, W^f_1 \in \mathbb{R}^{d_h \times d_f}, b^f_0 \in \mathbb{R}^{d_f}$ and $b^f_1 \in \mathbb{R}^{d_h}$ ($d_f = 4d_h$) are trainable matrices.

Finally, we can obtain the final location embedding h^N_i by N attention layer and the entire graph embedding $\overline{h}$ is:

$$\overline{h} = \frac{1}{n+1} \sum_{i=0}^{n} h^N_i \tag{16}$$

2) Decoder: For each decoding step, we convert the current traffic condition (i.e., the traveling time from last visited node i in the partial route to other nodes at time t) and the visited status of locations to the current traffic embedding $w^{(t)}$ and visited status embedding $v^{(t)}$, respectively.

We consider the embeddings h_0, h_l, $w^{(t)}$, $v^{(t)}$ (h_l denotes the feature of last visited location in the partial route) as the representation of the context vector C_t at time t, which is very informative for selecting the next node. We utilize GRU to cope with such sequential information, as it contains less parameters than LSTM and easier to train in the decoder.

$$H_t = GRU(C_t, H_{t-1}) \tag{17}$$

The entire graph embedding runs through the whole decoding process, so we set it as the initial hidden state of GRU (i.e., $H_0 = \overline{h}$). The output H_t of GRU is used as an enhanced context vector, which is treated as a query and compared with the key of each unvisited location based on the attention mechanism. Finally, the decoder selects the next unvisited location sampling from the output of a softmax layer.

To be more specific, suppose that the current location is i and the current time is t, the next visited location j can be determined by sampling on the probability distribution $p_0, \ldots, p_n$. Then the next timestep can be obtained as $max(E_j, t + \tau_{ij}(t))$. If a time window violation occurs (i.e., $t + \tau_{ij}(t) > L_j$), a penalty C_2 is added to the final cost. If the violation is the first one, C_1 is also added. Thereafter, the next decoding step can be proceeded until all the locations are successfully visited.

V. Experiments

The proposed method was implemented with PyTorch. All the experiments were conducted on a workstation with Intel(R) Xeon(R) CPU E5-2640 v4 at 2.40GHz and a single GTX1080Ti GPU.

A. Benchmark Dataset

To test the proposed algorithm and compare it with other methods, we make use of the real-world benchmark dataset in Beijing.

The benchmark dataset is composed of two problem sets, denoted as TSPTDTW-20 (Set 1) and TSPTDTW-40 (Set 2). For TSPTDTW-20, the real-world instances are generated by randomly selecting 20 customer locations in 100 potential locations from *Baidu Map*, normalized into a unit square. The settings of TSPTDTW-40 are consistent with TSPTDTW-20, except that the problem scale is 40. For ease of data collection, we discretized the time horizon into a set $T_1 = 12$ of timesteps, and sent requests to *Baidu Map API* every two hours of a day for each pair of locations to get the traffic pattern.

For these two problem sets, the traveling time between location i and location j at timestep t, called traffic pattern d_{ijt}, is recorded. The real-time traffic condition $\tau_{ij}(t)$ of each location pair at any arbitrary time t can be fitted using the cubic spline interpolation (CSI) [20] on d_{ijt}.

As for the time windows of customer locations, we generated them with the following steps. First, we calculated the average traveling time AT required to traverse every customers on a large number of instances without time windows. Then we set the time windows $[E_i, L_i]$ of each customer i by the following equations.

$$\begin{aligned} dt_i &\sim U(0, AT) + \alpha \\ len_i &\sim U(0, \beta) + \alpha \\ L_i &= \max(dt_i, len_i) \\ E_i &= L_i - len_i \end{aligned}$$

where $U(a, b)$ gives the uniform distribution between range $[a, b]$, dt_i denotes the expected deadline time of customer i, len_i denotes the length of time window. α and β are the controlled parameters determined the compactness of time windows. In our experiments, $AT = 1080$ and

TABLE I

Computational results of the objective cost and the number of problems feasibly solved on 10, 000 randomly generated instances on Set 1. (G): greedy decoding, (S-n): sample n solutions with the highest solution quality.

Method	TDTSPTW-20 (AT = 1080)							
	P72_144		P144_144		P144_288		P288_288	
	Cost ↓	Solved ↑	Cost ↓	Solved ↑	Cost ↓	Solved ↑	Cost ↓	Solved ↑
GR	8795.99	1342	4172.58	5861	3080.70	6941	175.22	9827
SA	8544.59	1561	3677.08	6343	2711.69	7301	133.69	9827
ID-AM(G)	8667.55	1469	3714.50	6331	2805.41	7218	120.33	9861
Traffic-AM(G)	10082.23	83	3840.33	6210	2852.64	7184	119.39	9866
LSTM-AM$^-$(G)	8714.42	1447	3825.42	6228	2909.05	7130	119.01	9871
MLP-AM(G)	8594.61	1547	3636.87	6404	2702.74	7329	118.76	9868
LSTM-AM(G)	8591.01	1555	3611.16	6427	2701.14	7327	114.61	9887
RNN^2-AM(G)	**8564.35**	**1586**	**3596.67**	**6445**	**2696.65**	**7335**	**102.67**	**9899**
RNN^2-AM(S-128)	8348.42	1753	3214.26	6808	2322.15	7694	57.97	9943
RNN^2-AM(S-256)	8321.15	1777	3179.76	6841	2305.27	7710	51.93	9949
RNN^2-AM(S-512)	**8290.19**	**1805**	**3146.55**	**6873**	**2280.25**	**7734**	**51.89**	**9949**

TABLE II

Computational results of the objective cost and the number of problems feasibly solved on 10, 000 randomly generated instances on Set 2. (G): greedy decoding, (S-n): sample n solutions with the highest solution quality.

Method	TDTSPTW-40 (AT = 1728)							
	P144_144		P144_216		P144_288		P288_288	
	Cost ↓	Solved ↑	Cost ↓	Solved ↑	Cost ↓	Solved ↑	Cost ↓	Solved ↑
GR	8807.02	1415	7945.35	2227	7219.83	2921	1123.41	8889
SA	**8640.26**	**1550**	7764.10	2382	7017.36	3100	1028.49	8981
RNN^2-AM(G)	8928.17	1281	**7678.04**	**2475**	**7010.18**	**3128**	**726.52**	**9289**
RNN^2-AM(S-128)	8197.30	1906	6484.63	3578	5719.47	4334	368.04	9638
RNN^2-AM(S-256)	8135.76	1962	6425.48	3633	5639.91	4410	353.71	9652
RNN^2-AM(S-512)	**8080.89**	**2012**	**6351.08**	**3704**	**5544.77**	**4502**	**340.42**	**9665**

$(\alpha, \beta) \in \{(72, 144), (144, 144), (144, 288), (288, 288)\}$ for Set 1. $AT = 1728$ and $(\alpha, \beta) \in \{(144, 144), (144, 216), (144, 288), (288, 288)\}$ for Set 2.

B. Baseline Algorithms

We consider *Greedy algorithm* (GR) and *Simulated Annealing algorithm* (SA) as the compared baseline algorithms. The procedure of GR is as follows. First, sort the locations according to the end time of the time windows from small to large. Next, for each sorted location, iteratively insert it into the partial route of the salesman, and eventually find a closed route. SA is a well-known and widely used heuristic algorithm. Five operators, including relocate, 2-opt, 3-opt, exchange and Or-opt [21], were adopted to improve the TDTSPTW solution, which was initialized by GR.

The reasons of choosing these two baselines are as follows. On one hand, none of them is heavily relied on the prior expert knowledge. These algorithms can be easily applied to other related applications with minor changes. On other hand, although these algorithms are slower than our method, they still require less computational time than other complex meta-heuristics and exact methods.

To further test our model, we also conducted ablation studies with modifications of **RNN^2-AM**, leading to **LSTM-AM**, **MLP-AM**, **LSTM-AM$^-$**, **ID-AM** and **Traffic-AM**. **LSTM-AM** discards the GRU module in the decoder. The following models are modified based on **LSTM-AM**. **MLP-AM** simply replaces the LSTM module with an MLP module. **LSTM-AM$^-$** ignores the traffic condition $w_i^{(t)}$ in the decoder. **ID-AM** ignores the traffic information in the encoder. **Traffic-AM** ignores the ID information in the encoder.

Finally, there are two different strategies in the decoding process: greedy and sampling. Greedy decoding means to select the node with the maximum probability, while sampling decoding means to sample the node according to the probability distribution.

C. Computational Results and Discussions

The training process of all the models was identical and presented in Algorithm 1. For each model, it was trained with parameters $E = 1,000$, $S = 512,000$ and $E_R = 100$. In the model testing, we report the average performance on 10, 000 test instances for each dataset.

To evaluate the performances (i.e., the objective cost and problems feasibly solved) of the proposed method, some TDTSPTW instance groups (P72_144, P144_144, P144_288, P288_288) with AT = 1080 on Set 1 and (P144_144, P144_216, P144_288, P288_288) with AT = 1728 on Set 2 were designed for the testing. Note the trailing two numbers represent the corresponding α and β values.

TABLE I and TABLE II present the results of compared methods on Set 1 and Set 2, respectively. From TABLE I, we can find that the proposed model **RNN^2-AM** outperforms the baseline algorithms in terms of the objective cost and the number of problems feasily solved on all TDTSPTW-20 tasks. Comparing RNN^2-AM with LSTM-AM, it shows that GRU module can produce enhanced context vector, which is very informative for selecting good nodes. Observing the results of **LSTM-AM** and **MLP-AM**, we note that LSTM module can perceive real-time traffic changes better than MLP module. This comparison verifies that LSTM module has a better ability to capture the temporal information. Similarly, we notice that our model can learn the traffic changes well by comparing **LSTM-AM** with **LSTM-AM$^-$**. The models **ID-AM** and **Traffic-AM** perform much worse than **MLP-AM** and **LSTM-AM**, which mean that both location and traffic information are important features. With the sampling decoding, our method significantly outperforms all the methods. As for the comparison of running time of a single instance in the testing phase, GR is about 3ms, SA is about 1.1s and each deep learning method is about 0.2s.

From TABLE II, **RNN^2-AM** clearly outperforms the baseline algorithms on almost all TDTSPTW-40 tasks except P144_144. However, our method can significantly surpass SA by sampling decoding on P144_144 task. As for the comparison of running time of a single instance in the testing phase, GR is about 1.7s, SA is about 7.5s and each deep learning method is about 0.4s.

VI. Conclusions

In this work, we propose a deep reinforcement learning approach to deal with the Time-Dependent Traveling Salesman Problem with Time Windows (TDTSPTW). To generate a satisfactory solution to the problem, we design an RNN^2-AM model to better perceive the real-time traffic information and produce enhanced context vector. Extensive experiments indicate that the proposed method is much more applicable in tackling TDTSPTW, compared with two typical baseline algorithms. It can capture the time-dependent features and yield a high qualified solution within a very short time. Ablation study shows that integrating the location information, traffic pattern with LSTM and context vector with GRU can help the model generate better decisions.

In future work, our method can be applied to solve other complex variants of dynamic routing problems (e.g., dynamic vehicle routing problem) with minor modifications. We only need to revise the encoder-decoder architecture to extract specific features of the studied problem.

Acknowledgements

This work is supported by the National Key R&D Program of China (No. 2018AAA0101203), the National Science Foundation of China (No. 71601191, 62072483) and Natural Science Foundation of Guangdong Province (No. 2019A1515011169, 2021A1515011301).

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.

[3] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *ICLR*, 2019.

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

[5] J. Schneider, "The time-dependent traveling salesman problem," *Physica A: Statistical Mechanics and its Applications*, vol. 314, no. 1-4, pp. 151–155, 2002.

[6] J. J. Schneider and S. Kirkpatrick, "Extensions of traveling salesman problem," *Stochastic Optimization*, pp. 233–241, 2006.

[7] P. A. Melgarejo, P. Laborie, and C. Solnon, "A time-dependent no-overlap constraint: Application to urban delivery problems," in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 2015, pp. 1–17.

[8] H.-B. Ban, "The hybridization of aco+ ga and rvns algorithm for solving the time-dependent traveling salesman problem," *Evolutionary Intelligence*, pp. 1–20, 2020.

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

[10] A. Arigliano, G. Ghiani, A. Grieco, E. Guerriero, and I. Plana, "Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm," *Discrete Applied Mathematics*, vol. 261, pp. 28–39, 2019.

[11] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[13] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NeurIPS*, 2015, pp. 2692–2700.

[14] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[15] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[16] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *NeurIPS*, 2018, pp. 9839–9849.

[17] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *ICLR*, 2018.

[18] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," *arXiv preprint arXiv:1911.04936*, 2018.

[19] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *NeurIPS*, 2018, pp. 539–548.

[20] S. A. Dyer and J. S. Dyer, "Cubic-spline interpolation. 1," *IEEE Instrumentation & Measurement Magazine*, vol. 4, no. 1, pp. 44–46, 2001.

[21] G. Babin, S. Deneault, and G. Laporte, "Improvements to the or-opt heuristic for the symmetric travelling salesman problem," *Journal of the Operational Research Society*, vol. 58, no. 3, pp. 402–407, 2007.