International Journal of Information Technologies and Systems Approach
Volume 17 • Issue 1

Logistics Distribution Route Optimization With Time Windows Based on Multi-Agent Deep Reinforcement Learning

Fahong Yu, Shanwei Institute of Technology, China*
https://orcid.org/0000-0002-9342-6419
Meijia Chen, Shanwei Institute of Technology, China
Xiaoyun Xia, Jiaxing University, China
Dongping Zhu, Shanwei Institute of Technology, China
Qiang Peng, Shanwei Institute of Technology, China
Kuibiao Deng, Shanwei Institute of Technology, China

ABSTRACT

Multi-depot vehicle routing problem with time windows (MDVRPTW) is a valuable practical issue in urban logistics. However, heuristic methods may fail to generate high-quality solutions for massive problems instantly. Thus, this article presents a novel reinforcement learning algorithm integrated with a multi-head attention mechanism and a local search strategy to solve the problem efficiently. The routing optimization was regarded as a vehicle tour generation process and an encoder-decoder was used to generate routes for vehicles departing from different depots iteratively. A multi-head attention strategy was employed for mining complex spatiotemporal correlations within time windows in the encoder. Then, a decoder with multi-agent was designed to generate solutions by optimizing reward and observing transition state. Meanwhile, a local search strategy was employed to improve the quality of solutions. The experiments results demonstrate that the proposed method can significantly outperform traditional methods in effectiveness and robustness.

KEYWORDS

Deep Reinforcement Learning, Logistics Distribution, Multi-Depot, Route Optimization

INTRODUCTION

With the rapid development of the transportation industry, more stringent requirements on vehicle routing have become an emerging issue in transportation service. This challenges vehicle management intelligence. Vehicle Routing Problems (VRPs) have been a subject of extensive research and attention

DOI: 10.4018/IJITSA.342084
*Corresponding Author

by scholars worldwide since its inception. To generalize the problem to a wider range of use cases, VRPs have been extended to include more complicated scenarios for a real-life environment. The fixed number of vehicles in the fleet and tighter time windows for customer demand have transformed traditional VRPs into vehicle routing problem with time windows (VRPTW). The problem can be characterized as the multi-depot vehicle routing problem with time windows (MDVRPTW). As a much simpler case, the multi-depot vehicle routing problem (MDVRP) itself is NP-hard, implying that it is unrealistic to generate optimal solution for a large-scale problem unless $P = NP$ (Braekers & Nieuwenhuyse, 2020).

For such extensively complicated problems, heuristic methods are conventionally considered as the viable solution tools. However, the logistics industry has been faced with a new challenge for serving massive amounts of requests instantly in the past few decades. Although many researchers propose diverse heuristics to solve the VRPs, it is still a challenging problem to provide reliable solutions for city scale problems within an acceptable amount of time. Artificial intelligence methods have been gradually evolving to tackle vehicle routing problems. Deep reinforcement learning (DRL) has become increasingly prominent in solving complex sequence decision problems, such as dynamic routing choice, automated vehicle control, and emergency evacuation. DRL is the fusion between reinforcement learning (RL) and deep learning (DL) which can addresses the issue of extreme large spaces with the action and state effectively. Subsequently, several studies attempted to solve the VRPs using DRL, with the encoder-decoder architecture being a popular choice for neural network design. Among these studies, an improved pointer network by simplifying the recurrent neural network (RNN) based encoder was proposed which resulted in more efficient solutions to the VRPs (James J., et al. 2019). With respect to graphic representations, an improved DRL incorporated with graph embedding network was given to solve the VRPs problem (Luis et al., 2019). In this model, VRPs were regarded as the route decoder process, and chose action in terms of the output of the graph embedding network. Moreover, a multi-agent attention model to solve the multiple vehicle routing problem with time windows was proposed (Zhang et al., 2020), which could achieve superior performance to several classical heuristic baselines with negligible calculating time.

Even though the proposed approaches have demonstrated superior performance to conventional methods in solving VRPs, most studies tend to focus on addressing straightforward routing issues that are essentially linear programming problems. Nevertheless, the MDVRPTW with various constraints is more challenging to solve. These constraints add complexity to the problem, making MDVRPTW more challenging to solve than traditional VRPs.

(1) The quality of heuristic methods is often determined by the quality of the groupings, and devising grouping rules requires a substantial amount of expert domain knowledge, making it difficult to achieve optimal results.
(2) Current research on DRL methods in combinatorial optimization problems mainly focuses on using a single agent to interact with the environment to solve problems like TSP and VRPs, while research on solving MDVRPTW is relatively lacking.
(3) Compared with single depot vehicle routing problems, the search efficiency of reinforcement learning will be compromised greatly for larger solution space of MDVRP. Furthermore, the decoder framework fixes the order of vehicles in transformer structure, which leads to the restrictions during the exploration of agents and is no longer effective to handle vehicles originated from different depots.

This paper proposed a multi-agent deep reinforcement learning with local search strategy (MADRLL) for MDVRPTW, which considers constraints such as multiple depots, multiple vehicles, and time windows. Then, a multi-head attention strategy was proposed to mine the complex spatiotemporal correlation encoder model within the time window, and a decoder with multi-agent was designed to generate solutions by optimizing rewards and observing transition

states. Meanwhile, a local search strategy to improve the quality of the solution was adopted to enhance overall performance.

The remainder of the paper is structured as follows. Related works are discussed in Section 2, and problems definition and mathematical modeling are presented in Section 3. Section 4 provides the description of the proposed algorithm. Section 5 presents the experiment results and analysis, and the paper is concluded in Section 6.

RELATED WORK

The solution strategies for VRPs mainly focus on heuristic methods. Although exact algorithms can obtain the optimal solutions, it was difficult to find the result in a reasonable time with the increases of problem scale and the computational complexity. VRPs can be solved by heuristic strategies by designing hand-crafted features with expert knowledge. The local search strategies were an important category of heuristic algorithms which can improve the solution quality. There were varieties of heuristic methods derived including simulated annealing (Hiermann et al., 2022), estimation of distribution algorithm (Wang, Li, & Guan, 2023) and tabu search (Gu et al., 2023). Local search strategies have been adopted in solving VRP, in which the search procedure stopped when the improvement of solutions in the neighborhood space were taking place again, while a premature convergence might be taking place (Stodola & Nohel, 2023).

Reinforcement learning was used to search the best solutions by maximizing accumulated rewards which were suitable for decision-making. The iterative process of RL to select a node is modeled as the following Markov Decision Process (MDP). (1) The agent chooses an action At according to the strategy π in the current state St. *(2)* The state St of the environment was transferred to the next state $St+1$ according to At. (3) The agent obtained the feedback reward Rt by the environment and chooses the next action $At+1$ according to strategy π. Common RL so*lv*es the decision strategy by iterating the Bellman equation, usually, however, the model was too costly. RL can be applied to sequence decision by a reward feedback strategy without requirement of labeled data. The difficulty of data annotation in the network and the network transmission performance used as natural decision feedback make DRL suitable for routing decisions.

With the increasing popularity of artificial intelligence, the research of DRL for VRPs was studied. DRL could be applied to automatically generate the road trajectory for the rescue vehicle in case of an emergency on the mountain freeway. A transformer-based architecture for the VRP was proposed (Li et al., 2021), where both the encoder and decoder of the neural network use the attention mechanism, significantly improving the computational speed and solution quality. In addition, DRL was used solve more complex VRPs, e.g., the heterogeneous capacitated vehicle routing problem, and the pairing and precedence relationships in the pickup and delivery problem (Lin et al., 2021). An end-to-end deep reinforcement learning framework to solve the EV routing problem with time windows (Ren et al., 2022), while the uncertain demands of customers in the VRP using DRL was addressed (Pan et al., 2023), resulting in efficient computation. A summary of this part of the study reveals that the existing studies did not consider the difference between the time window dimension and the logistics delivery time dimension. Therefore, the lack of variation between the two dimensions does not further exploit the potential for improving the solution quality.

It can be found that the existing research lacks the flexibility to search between different dimensions in the search space and knowledge exchange between agents, which inhibits further improvement of the algorithms. The multi-agent reinforcement learning framework can enhance the solution quality to some extent.

PROBLEMS DEFINITION AND MATHEMATICAL MODELING

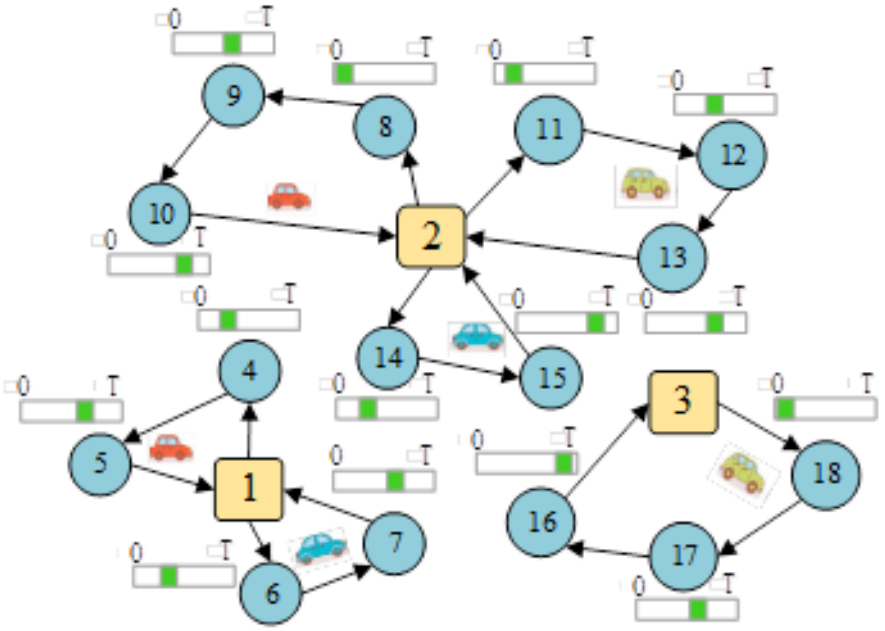
MDVRPTW

There are many branching variants of VRPs as the focus of in-depth studies, which have gradually expanded and extended into various real-life scenarios (Li et al., 2023). The fixed number of vehicles in the fleet and tighter time windows for customer demand have transformed traditional VRPs into VRPTW. Considering that vehicles depart from multiple depots, VRPTW was transformed into MDVRPTW which has more applications in real life by assigning limited vehicles with several trips for meeting the demands at the specified time. The appropriate penalties were incorporated into the reward to ensure the vehicle can satisfy the customer's demand within the time window. There were three depots {v1, v2, v3} and 15 customers {v4, v5, …, *v18}* used to demonstrate the MDVRPTW clearly in Fig. 1.

MDVRPTW involves multiple depots for delivering goods to multiple customers, and every depot is assigned a certain number of vehicles. The maximum capacity of each vehicle and the coordinate of every depot were known. The demand volume and location coordinates are known for every customer point, and every customer node has a specific time window and demand to be met. Vehicles need to satisfy customer demands within specified time windows. Customer demand quantities are fixed and come with time window requirements. The goal is to minimize the total route length by optimizing vehicle routes to satisfy customer requirements and time window constraints. There are a set of vehicles with capacity Q starts from multiple depots to serve all customer requests under its soft time windows. The assumptions are made for this problem as following:

(1) All vehicles are the same type, and their maximum capacities are identical.
(2) Each customer is serviced by exactly one depot.

Figure 1. An overview of the problem scenario



(3) The demand quantity of every customer is less than the maximum capacity of every vehicle. The time window of each customer is unambiguous. The penalties must be paid if the vehicle does not arrive during the time window.
(4) The coordinate of every depot and every customer is known clearly. The depot of departure and return of each vehicle is the same.
(5) There are no transportation routes between depots.

Mathematical Modelling

This paper focuses on solving any random instance which is the symmetric two-dimensional Euclidean MDVRPTW and formulated as an undirected graph *G*=(*V*, *E*) where there are nodes including depots and customers. In the graph, V = D*ep*∪*Cus*={*v1,..., vD*} ∪{*vD+1,..., vD+N*}. Here Dep = {*v1,..., vD*} and *Cus* = {*vD+1,..., vD+N*} represents a set of depots and customers respectively, and *E* ={*eij* | *D+1*≤*i*≤*D+N, D+1*≤ *j*≤*D+N*} denotes a set of edges, *e*ij= di*s(i, j)* means distance between node v*i* and v*j*. A nod*e* vi is ass*oc*iated with a list $X_i = \left(x_i, y_i, e_i, l_i, q_i, \beta_i^e, \beta_i^l\right)$ where $\left(x_i, y_i\right)$ represent the coordinate of node vi, $\left[e_i, l_i\right]$ represent the time window from ei to l_i, q_i is the demand of customer i, and β_i^e, β_i^l represent early punishment coefficient and late punishment coefficient, respectively. The customer's demand will be produced by sampling randomly in this paper. For those nodes Dep = {*v1,.. , vD*}, the demand of each depot is set to 0 and *d* time window of each depot is set to [0, *T*] allowing that the depots do not require any demand and it can be accessed by vehicles at any time.

Table 1 summarizes the notations adopted to define the problem. Given π as the solution sequence, the goal is to minimize the total routing length τ.

MDVRPTW allows some flexibility in the arrival time compared to customer service times. However, penalties are imposed for arriving late or early, and these penalties are added to the reward to encourage vehicles to meet customer demands within the specified time windows. The set of vehicles is {*Md* | ∀*d*∈*Dep*}, M*d* is the set of vehicles in depot d. Considering every vehicle is dedicated to a unique route, a total number of M (that is $\sum_{m=1}^{D}\left|v_m\right|$) routes will be generated, and they only connect to each other at the depot. The max capacity of each vehicle is Q and the residual capacity of the *m*th vehicle at timestamp *t* is $\hat{q}_m^t$. The cost is measured by Euclidean distances in the plane, and the speeds of all vehicles are assumed to be identical. *Dis(i, j) = SQRT((*$x_i - x_j$*)*^{2} *+ (*$y_i - y_j$*)*^{2}*), ∀i∈V, ∀j∈V)*. An element of time-variant η_i is the system time at decoding step *t* for vehicle *i*, which is set to *0* as initial value. The optimization solution is to find the optimal permutation $\pi = (\pi 1, \ldots, \pi n)$ with minimal total travel cost of all vehicles τsum*(π1..M) and* total punishment for violating constraints psum*(π1..M) of* time window, which were defined as Eqs. (1) and (2). The elements $\pi t \in V$ *in per*mutation π selected at each time step $t \in \{1, \ldots, n\}$ *are the o*rders of those nodes in the graph. Feasible permutation π must satisfy two conditions: (1) each customer is served exactly once; (2) all customers can only be served once, $\pi t \neq \pi t'$, $\forall t \neq t'$.

$$\tau_{sum}\left(\pi_{1..M}\right) = \sum_{d=1}^{D}\sum_{m=1}^{M_d}\left(\sum_{i=1}^{|\pi_{d,m}|-1} dis(\pi_{d,m}[i], \pi_{d,m}[i+1]) + dis(\pi_{d,m}[|\ \pi_{d,m}\ |], \pi_{d,m}[0])\right) \quad (1)$$

$$p_{sum}\left(\pi_{1..M}\right) = \sum_{d=1}^{D}\sum_{m=1}^{M_d}\sum_{i=1}^{|\pi_{d,m}|-1}\left[\beta_i^e \max\{(e_i - \eta_i), 0\} + \beta_i^l \max\{(\eta_i - l_i), 0\}\right] \quad (2)$$

where dis(·,·) is *the Eucl*idean distances in the plane, τd,m[t] means the *c*ustomer served by the mth vehicle *s*tarted out from dth depot at timestamp *t*. The early punishment coefficient $\beta_i^e < 0$ and late punishment coefficient $\beta_i^l < 0$.

Table 1. The variable and parameter definitions of the MDVRPTW model

Variable	*Description*
$V=\{v_0, v_1, \ldots\}$	*Node set*
D	*Number of Depots*
v_j	*Node of Depot j, j={1, …, D}*
v_i	*Node of customer i, i={D+1,…, D+N}*
n	*Number of customers*
$E=\{(v_i, v_j)\mid i \neq j\}$	*Edge set*
(x_i, y_i)	Vector representing coordinate of node v_i
$[e_i, l_i]$	Time window from e_i to l_i
Q	*max capacity of each vehicle*
$\hat{q}_m^t$	*The residual capacity of the mth vehicle at timestamp t*
q_i	*demand of customer i*
T	*the whole-time interval*
$dis(i,j)$	*distance between v_i and v_j*
π	*solution sequence*
η_i	*the system time of the vehicle i*
τ	*total routing length*

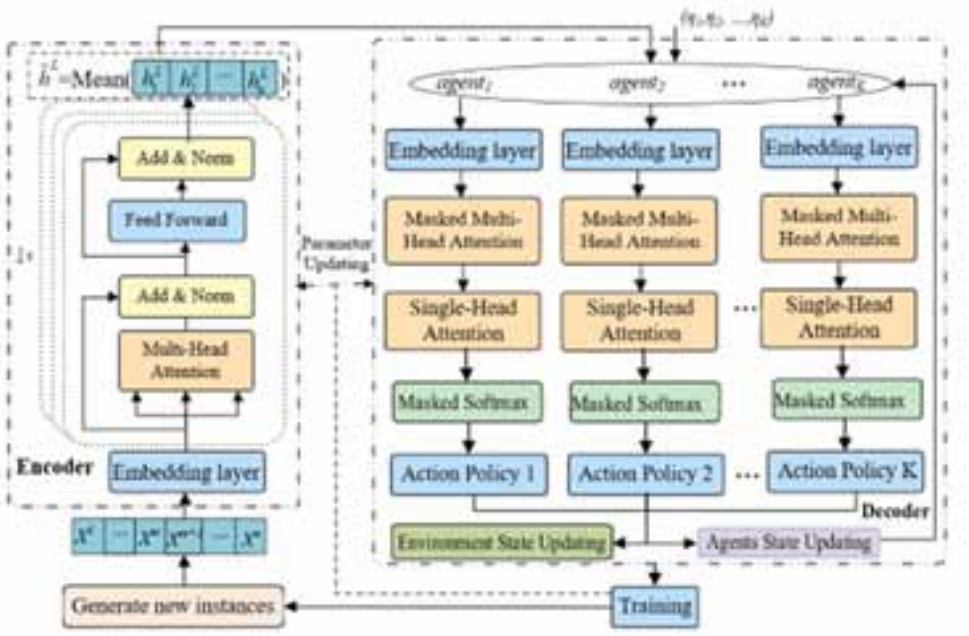
Algorithm Description

The framework of the proposed methodology can be described as Fig. 2 which produces the routes as a series of consecutive actions by the architecture composed of inputs, an encoder, and a decoder comprising multiple agent networks. In this architecture, the main objective of using the encoder is to understand the information of depots and customers through creating a high-dimensional representation by extracting spatial-temporal information within the time window network. The representation will be forwarded to the decoder to generate a sequence of its own that represents the output.

Parameterized stochastic policies are used at every time step t, and actions are selected based on the probability vector output by the policy network until the end state is reached (i.e., all customer points have been visited). The final output of the policy is a complete node selection sequence, denoted as $\pi=\{\pi_1, \pi_2, \ldots \pi_T\}$, where T is the length of the selected node sequence. According to the chain rule, given an instance s, a stochastic policy $p(\pi|s)$ was parameterized by θ for *s*electing a solution $\pi_{1..M}$ as shown in Eq. (3).

$$P\left(\pi \mid s\right)=\prod_{t=1}^{T} p_{\theta}\left(\pi_{t} \mid s_{t-1}, \pi_{t-1}\right) \tag{3}$$

Figure 2. Framework of DRL algorithm



where T denotes the step, $p_\theta(\pi_t|s_{t-1}, \pi_{t-1})$ indicates the probability of selecting customer at step t. θ represents parameters to be learned.

Encoder Architecture

The encoder embeds the raw features of a problem instance (i.e., customer location, customer demand, and vehicle capacity) into a higher-dimensional space, and then processes them through attention layers for better feature extraction. The encoder is composed of an embedding layer and N attention modules with identical structures but independently parameterized. Each attention module consists of a multi-head attention layer (MHA) and a feed-forward layer (FF). The input to these modules is the feature of every node, and the output is a higher-level feature representation of every node and the overall graph feature information.

Step 1: Embedding Layer. The embedding layer takes the features of depots and customer points as inputs, where $X=\{X_i|\forall i \in V\}$, and $X_i = \left(x_i, y_i, e_i, l_i, q_i, \beta_i^e, \beta_i^l\right)$. Where x_i, y_i represent the coordinates of the node in a *2D* plane, e_i and l_i represent the time window, and q_i is the demand of customer i. The demand of depot is set to *0*. The embedding layer maps every input X_i to a node embedding feature h_i^0 (with feature dimension dim(h_i^0) = 128), as shown in Eq. (4).

$$h_i^0 = W^X \times X_i + b^X \; i \in \{1,..., D, D+1,..., D+N\} \tag{4}$$

where W^X and b^X represents the learnable parameters of embedded layer.

Step 2: Attention Modules. The initial input to every attention module is the embedding feature h^0. Each module consists of MHA and FF layers, which both use residual connections and batch-normalization. The attention module updates the node features from the previous layer, h^{l-1} to h^l, where $l \in [1,N]$, representing the l-th attention module, as shown in Eqs. (5) and (6).

$$\hat{h}^{l} = BatchNorm^{l}\left(h^{l-1} + MHA^{l-1}\right) \tag{5}$$

$$h^{l} = BatchNorm^{l}\left(\hat{h}^{l} + FF^{l}(\hat{h}^{l})\right) \tag{6}$$

Step 2.1: Multi-Head Attention Layer. Attention mechanism is to learn a probability distribution that can essentially be described as the process of mapping a query vector, key vector, and value vector to an output vector by connecting with context. The output vector is the weighted sum of values, which can be calculated according to Eq. (7).

$$Attention_{(Q,K,V)} = SoftMax(\frac{QK^{T}}{\sqrt{d_{k}}})V \tag{7}$$

Multi Head Attention (MHA) is a variant integrated multiple head of attention mechanisms. Each head can learn different information from different representation spaces, which will contribute to capturing the dependency between input and output.

By calculating the attention distribution, the degree of attention to a certain input can be obtained. There are three matrices Q, K and V calculated by multiplying the input vector H with W^{Q}, W^{K}, and W^{V} which are three weight matrices by initializing randomly. In this work, attention heads $M = 8$ was adopted to compute attention in *8* different sub-spaces, every with dimensions of $dim(h) / M = 16$. It can be calculated as Eqs. (8)-(10).

$$q_{m} = W_{m}^{Q} \times h^{l-1} \tag{8}$$

$$k_{m} = W_{m}^{k} \times h^{l-1} \tag{9}$$

$$v_{m} = W_{m}^{v} \times h^{l-1} \tag{10}$$

Where q_{m}, $\mathrm{k_m}$ and v_{m} are query, key, and value for the *m-th* dimension space, and W_{m}^{Q}, W_{m}^{K}, W_{m}^{v} are the corresponding network parameters.

To account for specific characteristics of MDVRPTW, the definition of node neighbors is revised. If a node is a depot, its neighboring nodes correspond to all customer points. If a node is a customer, its neighboring nodes include all nodes. In every attention head's dimension space, the scaled dot-product value u_{ij}^{m} between $q_{i,m}$ and $k_{j,m}$ is calculated. It is then normalized using the *SoftMax* function to obtain the attention score $u_{ij}^{m} \in [0,1]$. The attention score a_{i*}^{m} is used to compute the dot product with the corresponding $v_{i,m}$, resulting in feature sub-space for every attention head. Finally, all feature sub-spaces from different attention heads are merged to form a complete node feature. It can be calculated as Eqs. (11) and (12).

$$u_{ij}^{m} = \begin{cases} \dfrac{q_{i,m}^{T} \times k_{j,m}}{\sqrt{\dim(k_{j,m})}}, & j \text{ adjacent to } i \\ -\infty, & otherwise. \end{cases} \tag{11}$$

$$a_{ij}^{m} = SoftMax(u_{ij}^{m}) = \frac{e^{u_{ij}^{m}}}{\sum_{j'} e^{u_{ij'}^{m}}} \tag{12}$$

The attention score u_{ij}^{m} computes based on a scaling mechanism. Due to the large scalar results generated by inner product operation, the SoftMax function falls into the saturation region with a small gradient that may occur gradient dispersion during the training process. The output of the attention mechanism can be obtained by weighting evaluation with value as Eq. (13).

$$h'_{i,m} = \sum_{j'} a_{ij}^{m} \times v_{j}^{m} \tag{13}$$

Then the output of multi-head attention will be concatenated by attention outputs from *M* subspaces, which can be calculated as Eq. (14).

$$MHA(h_i) = \sum_{m=1}^{M} W_{m}^{O} \times h'_{i,m} \tag{14}$$

Where W_{m}^{O} is feature fusion of network parameters for multi-head attention.

Step 2.2. Feed-Forward Layers. The feed-forward layer consists of two linear fully connected layers, and *ReLU* activation is applied to the neurons, as shown in Eq. (15).

$$FF(\widehat{h_i}) = W_{2}^{F} \times \mathrm{Re}\,LU(W_{1}^{F} \times \widehat{h_i} + b_{1}^{F}) + b_{2}^{F} \tag{15}$$

where *ReLu(x) = max(x,0)*, W_{1}^{F}, b_{1}^{F} are the parameters for the first fully connected layer, and W_{2}^{F}, b_{2}^{F} are the parameters for the second fully connected layer. $h_{i}^{(1)}$ indicates an output of one attention module. The final customer embedding $h_{i}^{(\lambda)}$ for every customer $i \in \{1, 2, \ldots, N\}$ can be obtained after λ attention modules.

In the circumstances with multiply depot, each vehicle was required to return to the depot where the vehicle starts from. The information for all customers and depot *j* is indicated by an aggregated embedding shown in Eq. (16).

$$\bar{h}_{j}^{(N)} = \frac{1}{N+D}\left(h_{j}^{(\lambda)} + \sum_{i=D+1}^{D+N} h_{i}^{(\lambda)}\right) \tag{16}$$

Decoder Architecture

Definition of Basic Elements for MDP

A Markov decision process can be constructed for MDVRPTW. The related elements contained agents, state space, actions, state transitions, and reward function and can also be designed to formulate a multi-agent reinforcement learning framework.

(1) Agents. Vehicles were regarded as agents which used to observe the environment and the states of other agents. Each agent selects an action based on the perceived information and influences the states at time step t.
(2) States. The state at step t contained two parts such as the agent state and environment state. state $S = \{S_g, S_a\}$ contains a global state S_g representing the static state, which is the overall graph feature information output by the encoder, and the agent state $S_a = \{S_{a,1}, S_{a,2}, \ldots, S_{a,M}\}$ consisting of the states of all agents. The state of an individual agent *m* belongs to the dynamic state $S_{a,m,t}$

$= \{ h^{(N)}_{m,\pi_{t-1}}, \hat{q}^t_m \}$ changing over time. Here, $h^{(N)}_{m,\pi_{t-1}}$ represents the node features chosen by agent m in the previous step. The residual capacity for vehicle m will be updated as $\hat{q}^t_m = q^{t-1}_m - q_i$ after serving customer i. The environmental state mainly contains the information of those unvisited customers.

(3) Actions. The action space in multi-agent reinforcement learning consists of the joint action space $A^t = \{ A^t_m \}$ for all agents, where $m= 1,2...., M$. The action A^t_m for agent m at the current time step t represents the choice of the node to be served by that agent, including unvisited customer points and the corresponding distribution center for that agent.

(4) State Transitions. After time step t, when agent m selects action A^t_m, the agent's state transitions to $S^t_{a,m} = \left\{ S^{t-1}_{a,m} * A^t_m \right\}$, where "*" indicates the addition of the selected node from the action to the current state, continuing until the complete joint action A^t is formed and the complete state transitions from S^{t-1} to S^t.

(5) Rewards. Designing a well-structured reward function is crucial for training agents. The objective is not just to maximize a single immediate reward but to maximize the long-term cumulative reward. The reward function is defined based on the sequence $\pi_{1..M}$ of M customer nodes decoded by the decoder. For the Vehicle Routing Problem with Time Windows (VRPTW), the goal is to minimize the route cost. Therefore, the term $\tau_{sum}(\pi_{1..M})$ is added to the reward function to encourage shorter distances, as larger rewards are associated with shorter distances. For MDVRP, the objective function is to minimize both the total distance cost and the time cost. A smaller total cost results in higher cumulative rewards for the agents. Therefore, the negative of the total cost is used as the cumulative reward, which was defined as Eq. (17).

$$R=\gamma\tau_{sum}(\pi_{1..M}) + p_{sum}(\pi_{1..M}) \tag{17}$$

In contrast to supervised learning, where model parameters are optimized based on the difference between labels and predictions, the reward function serves as a guide for updating the reinforcement learning policy network and value network. Reinforcement learning agents continually iterate to maximize the expected total future return based on these reward signals. The reward signal is designed to consider both total logistics delivery time cost and waiting time cost, where $\gamma < 0$.

Decoding Strategy

Contextual information $S^{t-1} = \{S_g, S^{t-1}_a\}$ from both the encoder and the feature embedding module at every step were applied to decode to obtain a probability vector for selecting every customer. The specific network structure of the decoder consists of multiple agent networks and every agent network consists of an embedding layer, a masked multi-head attention layer and a single-head attention layer. For the scenario with multiple depots, it is necessary for each vehicle to return to the depot that it is originating from. A matching strategy takes the choice of both vehicles and customers into account, which allows larger exploration space for the vehicles belonging to different depots. During the course of the decoder, the agent and customer may be chosen based on the state information of each vehicle and the customer embedding.

Step 1. Embedding Layer. The embedding layer maps the state feature h^c_m of the agent m to the context-aware vector $\hat{q}^c_m$ where h^c_m includes overall graph feature information $\tilde{h}^L$, current agent features $\{ h^{(N,\ d)}_{m,\pi_t}, \hat{q}^t_m \}$, and the features $\{ h^{(N,o)}_{i,\pi_t}, \hat{q}^t_i \}$ of other agents. To commence decoding, the decoder must obtain contextual information including the graph embedding, the embedding of the preceding node, and the feature embedding. The main task of the feature embedding is to aggregate

customers information, information of current agents, and information of the other agents, which can be computed as Eq. (18).

$$h_m^c = concat\left(\tilde{h}^L, h_{m,\pi_{t-1}}^{(N,d)}, \hat{q}_m^t, \left\{h_{i,\pi_{t-1}}^{(N,o)}, \hat{q}_i^t\right\}\right) m \in \{1,2,\ldots,M\},\ i = 1,2,\ldots,M\ (i \neq m) \tag{18}$$

Where d represents the start depot d of vehicle m and o represent the start depot o of the other vehicle i, in which d and o are the same depot if they depart from the same depot.

Step 2: Masked Multi-Head Attention Layer. The MHA in the decoder is slightly different from that in the encoder, in which those customers have been traveled will be hid (set to -∞) before the softMax operation is carried out. The argument query in MHA can be obtained by calculating from the context-aware vector $\hat{q}_m^c$ in the embedding layer in the decoder, which can be calculated as Eq. (19). The argument's key and value in MHA can be produced by calculating the eigenvalue of node from the output of encoder, which can be calculated as Eqs. (20) and (21).

$$\hat{q}_m^c = W^C \times h_m^c + b^c \tag{19}$$
$$\hat{k}_{m,i} = \hat{W}_{m,i}^K \times h^N \tag{20}$$
$$\hat{v}_{m,i} = \hat{W}_{m,i}^V \times h^N \tag{21}$$

Where W^C and b^C are the network parameters of the decoder embedding layer. $\hat{W}_{m,i}^K$, $\hat{W}_{m,i}^V$ are the network parameters for computing the key and value respectively, $i = 1,2,\ldots,I$ is the dimensional space for every attention head, and m represents current agent.

Then the scaled dot product vector $\hat{u}_{m,j,d}^t$ between query and key can be calculated as Eq. (22). To satisfy the problem's constraints, a masking mechanism in MHA is applied to mask nodes (set $\hat{u}_{m,j,d}^t$ to $-\infty$) if any one of the following conditions is not satisfied. (i) The node j is a neighboring node of the current agent corresponding to the depot d. (ii) The node is unvisited. (iii) The demand of the node j is smaller than the remaining capacity of the current agent corresponding to the vehicle.

$$\hat{u}_{m,j,d}^t = \begin{cases} C \times \tanh\left(\dfrac{(\hat{q}_m^c)^T \times \hat{k}_{m,j,d}}{\sqrt{\dim(\hat{k}_{m,j,d})}}\right), & under\ conditions\ (i)\text{-}(iii) \\ -\infty, & otherwise. \end{cases} \tag{22}$$

where C means to clip the compatibility within interval. $C \in [-10, 10]$, m, j and d represent the current agent, the customers, and original depot, respectively.

Then the normalized attention score $\hat{\alpha}_{m,i}$ can be acquired using the SoftMax function as shown in Eq. (23), and the aggregation of the MHA can obtain the latest context-sensitive vectors q_m^c as Eqs. (23) and (24).

$$\hat{\alpha}_{m,j,i} = SoftMax\left(\hat{u}_{m,j,i}\right) = e^{\hat{u}_{m,j,i}} \Big/ \sum_{j'} e^{\hat{u}_{m,j',i}} \tag{23}$$

$$\hat{q}_{m,i}^c = \sum_j \hat{\alpha}_{m,j,i} \times \hat{v}_{m,j,i} \tag{24}$$

$$q_m^c = \sum_{i=1}^{I} W_{m,i}^O \times \hat{q}_{m,i}^c \tag{25}$$

Where $W_{m,i}^{O}$ represents the network parameters aggregated by the feature of multi-head attention in decoders.

Step 3. Single-Head Attention Layer. The action probability vector for node selection can be produced by a single-head attention layer in decoder. The compatibility $u_{m,j}$ between query and key in the single-head attention layer can be obtained as Eq. (26). The query can be calculated from the context-dependent vector q_m^c from the multi-head attention layer and the key can be calculated according to equation $k_{m,j} = W_m^K \times h_j^N$.

$$u_{m,j} = \begin{cases} C \times \tanh\left(\dfrac{(q_m^c)^T \times k_{m,j}}{\sqrt{\dim(k_{m,j})}}\right), & under\ conditions\ (i)\text{-}(iii) \\ -\infty, & otherwise. \end{cases} \tag{26}$$

The selection probability $p_{m,j}$ for every node is obtained by normalization by SoftMax function as Eq. (27). Action selection for every agent is based on the probabilities provided by the attention layer.

$$p_{m,j} = SoftMax\left(u_{m,j}\right) = e^{u_{m,j}} \Big/ \sum_{j'} e^{u_{m,j'}} \tag{27}$$

Where $p_{m,j} = \left(e^{x} - e^{-x}\right) \Big/ \left(e^{x} + e^{-x}\right)$ is the hyperbolic tangent activation function. C means to clip the compatibility within interval. $C \in [-10, 10]$, m and j represent the current agent and the customers, respectively.

The entire decoding process, involving the action selection and policy, is iteratively executed. According to the probability vector p_m outputted by each agent, the action selection policy selects the next action A_m^t, of that intelligence, and obtains the joint action $A^t = \left\{A_1^t, A_2^t, \ldots, A_M^t\right\}$, until all the customers have visited and forms the complete policy solution $\pi = \left\{\pi_1, \pi_2, \ldots, \pi_T\right\}$.

Policy Network Training Method

Given the high demand for labels in policy network training, here the REINFORCE algorithm has been employed with a rollout baseline (Williams, 1992) as a basic architecture to train a multi-agent joint policy network model. This is done by estimating the policy gradient of the joint policy using cumulative returns and training the multi-agent policy network. For a given instance s, the policy network θ outputs action probability vectors for every step of all agents, denoted as $p_\theta(\pi_t|s)$. It then samples the joint policy $\pi_t = sample(p_\theta(\pi|s))$ using these probabilities. The baseline network θ^{bl}, on the other hand, outputs action probability vectors $p_{\theta^{bl}}(\pi_t \mid s)$ and selects the joint policy $\pi_t^{bl} = greedy\left(p_{\theta^{bl}}\left(\pi_t \mid s\right)\right)$. The expected cumulative reward $L\left(\theta|s\right) = E_{p_\theta(s)}\left[R\left(\pi\right)\right]$ of the policy is evaluated using the Monte Carlo algorithm, where $R(\pi)$ represents the cumulative return of the policy $\pi = \{\pi_1, \pi_2, \ldots, \pi_T\}$. The policy gradient is calculated using the REINFORCE algorithm with a baseline and policy network parameters are updated using gradient descent as shown in Eqs. (28) and (29).

$$\nabla_\theta L\left(\theta|s\right) = -E_{\pi \sim p_\theta(\cdot|s)}\left[R\left(\pi_{1..M}\right) - R\left(\pi_{1..M}^{bl}\right)\right] \times \nabla_\theta \log p_\theta\left(\pi_{1..M}|s\right) \tag{28}$$

$$\theta = Adam\left(\theta, \nabla_\theta L\left(\theta|s\right)\right) \tag{29}$$

where $R\left(\pi^{bl}_{1..M}\right)$ means the reward.

The variance of the gradient of training the network can be reduced effectively by evaluating instance *s* using the baseline network θ^{bl}. The baseline network is updated in a rolling manner. The parameters θ^{bl} of the baseline network will be replaced by parameter θ if the improvement is significant based on the experimental verification by paired *t-test* with a significance level of α *(= 0.05)* in the training of each epoch (Kingma, D. P & Ba, J. L., 2015). The framework of the policy gradient is demonstrated in Algorithm 1.

The policy gradient algorithm for policy training is depicted in Algorithm 1 which consists of the policy network and baseline network. The policy network generates a probability vector at each step and selects an action by sampling. In contrast, the baseline network shares the same structure as the policy network but chooses the action with the highest probability based on greedy policy. The gradient of the loss function is provided in Eq. (28). In each epoch, the parameters of the policy network are updated according to the gradient. At the same time, a t-test is performed on the current model. When the reward of the policy network is significantly better than that of the baseline network,

Algorithm 1. Policy gradient algorithm

Input:	*Initialization: Training instances set I; number of epochs MAXE; batch size B; number of batches MAXB=|I|/B; significance level δ; number of agents M; maximum decoding length T; Initial parameters θ for policy network π^θ; baseline policy network with trainable parameters θ^b for policy network π^b.*
Output	*trained parameter set θ*
1)	*Initialize θ, θ^b*
2)	*for epoch = 1, 2, …, MAXE do:*
3)	*for batch = 1, 2, …, MAXB do:*
4)	*s_i ← Sampleinput(I), ∀i∈{1,2,…,B}*
5)	*{ A_1, A_2,…, A_M} ← Selectstartdepots(D)*
6)	*Initialize the start time η_m and state S^m for all agents, ∀m∈{1,2,…,M}*
7)	*for t= 1, 2, …, T do:*
8)	$\hat{q}^c_m = concat(\tilde{h}^L,\ h^{(N)}_{m,\pi_{t-1}},\ \hat{q}^t_m, \{h^{(N)}_{m,\pi_{t-1}},\ \hat{q}^t_m\})$ *for each agent*
9)	$\left\{\pi^1_i,\ldots,\pi^M_i\right\} \leftarrow p_\theta\left(\pi^m_t \mid s_i, \pi^m_{1:t-1}\right)$, ∀ ∈{1,2,…,B}, ∀m∈{1,2,…,M}
10)	$\left\{\pi^{b,1}_i,\ldots,\pi^{b,M}_i\right\} \leftarrow p_{\theta^b}\left(\pi^m_t \mid s_i, \pi^m_{1:t-1}\right)$, ∀i∈{1,2,…,B}, ∀m∈{1,2,…,M}
11)	Observe reward $\{r^1_t, r^2_t, \ldots, r^M_t\}$ and next state s_{t+1};
12)	*end for*
13)	*if batch%5==0 then*
14)	*π* ←LS(π_i);*
15)	$b(s) = \min\left\{\left\{R\left(\pi^{b,m}_i\right)\right\} \cup R\left(\pi^*\right)\right\}$, ∀i∈{1,2,…B}, ∀m∈{1,2,…,M}}
16)	*else*
17)	$b(s) = \min\left\{R\left(\pi^{b,m}_i\right)\right\}$, ∀i∈{1,2,…B}, ∀m∈{1,2,…,M}}
18)	*end if*
19)	Compute $\nabla_\theta L\left(\theta|s\right)$ according to Eq. (28).
20)	Update θ according to Eq. (29).
21)	*end for*
22)	*if Paired t - test* $\left(p_\theta, p_{\theta^b}\right) < \delta$ *then*
23)	*θ^b ← θ*
24)	*end if*
	end for

the parameters of the baseline network are replaced with those of the policy network. Algorithm 2, a local search strategy, is applied. It will be conducted every few iterations so as to overcome the extraordinary waste of computing time in the process of local search.

Action Selection and Local Search Strategy

Action Selection Strategy

The policy network, after multiple rounds of learning, possesses good decision-making capabilities. The decision-making activities were carried out based on the probability vectors output by the network. A sampling strategy was applied in this paper based on the decoder's output probabilities as a sampling probability distribution. Therefore, this strategy does not always choose the action with the highest probability but selects actions with different probabilities.

During model training, the baseline network θ^{bl} acts as an evaluator for the difficulty of each batch of training instances. Using a greedy action selection strategy quickly obtains an effective "evaluation metric." The policy network θ, acting as the "actor," needs effective evaluation of its decision-making ability. The sampling action selection strategy effectively estimates the expected value of solution quality, representing the actor's decision-making ability. By having θ^{bl} and θ choose suitable actions using different strategies, the learning efficiency and model performance can be improved.

For all nodes i at each decoding step t, the probabilities p_i^t can be estimated and the agent can decode solutions for an MDVRPTW instance. Particularly, there are three decoding strategies as follows.

(1) Greedy Decoding: The customer with the highest probability at each step t was selected greedily as the next node, i.e. next node $j = argmax\left\{p_i^t\right\}$.

(2) Stochastic Sampling: The next node to visit according to the probability distribution described by p_i^t was sampled for all i, at each decoding step t.

The time complexity of greedy decoding was the lowest among these strategies, while the poor solutions may be generated because of its myopic nature and the lack of exploration in the solution space. So, a stochastic sampling strategy was applied to the train model.

Local Search Strategy

The trained MADRLL can quickly solve the MDVRPTW using a greedy action selection strategy. However, there are some requirements to improve for some challenging instances, such as issues with route crossings in sub-routes and overconfident behavior in greedy action selection. To improve solution quality, this paper proposes an improved local search strategy according to the characteristics of MDVRPTW. Due to the advantage of fast a solution, repeated sampling of the model is not very costly. The framework of local search strategy was demonstrated in Algorithm 2.

The local search strategy optimizes each sub-route and sample search to avoid over-confident behaviors of the policy network. The main idea of the proposed strategy is to randomly swap two customers of a given sub-route to generate a new one so that its cumulative reward is shorter than that of the original route. Given an original route $\{..., c_i, c_{i+1}, ..., c_j, c_{j+1}, ...\}$, a new route $\{..., c_i, c_j, ..., c_{i+1}, c_{j+1}, ...\}$ can be obtained by swapping the positions of customers c_j and c_{i+1} and the shorter route will be formed. In this strategy, the operation takes place for each sub-route S_k in possible solution set S. There are two integer $i \notin (0, len(S_k\text{-}1))$ and $j \notin (0, len(S_k\text{-}1))$ sampled randomly in condition $|j\text{-}(i+1)| \geq 2$. Two edges (c_i, c_{i+1}) and (c_j, c_{j+1}) are recombined into new edges (c_i, c_j) and cost (c_{i+1}, c_{j+1}). The original edges will be removed, and the new edges will be generated if the cost of two new edges is smaller than original edges. To overcome the extraordinary waste of computing time in the proposed strategy, it will be conducted every few iterations. Given that there must be

Algorithm 2. Local search strategy LS(S)

Input:	*possible solution set S*
Output	*route S**
1)	*for each sub-route S_k in S do: // S_k ={…, c_i, c_{i+1}, …, c_j, c_{j+1}, …}*
2)	*for iter = 1,…, len(S_k)/5 do:*
3)	*i, j ←SampleInt(len(S_k-1)) ∧ (|j-(i+1)| ≥ 2);*
4)	*if cost(c_i, c_j) + cost (c_{i+1}, c_{j+1}) < cost(c_i, c_{i+1}) + cost (c_j, c_{j+1}) then*
5)	*(c_i, c_{i+1}) ← (c_i, c_j), (c_j, c_{j+1}) ← (c_{i+1}, c_{j+1});*
6)	*end if*
7)	*end for*
8)	*end for*
9)	*S* ← S;*
10)	*return S*;*

some solutions with relatively poor quality after each iteration, only a small portion (20%) of the optimal routes will be selected randomly to implement the operation. The poor routes are replaced with the optimized routes.

EXPERIMENTAL RESULTS AND ANALYSIS

To test the performance of the proposed algorithm, a series of experiments were implemented. The overall framework of MADRLL is implemented under Pytorch environment. The training policy network model were executed on a RTX 2080Ti GPU, and the arithmetic cases were tested on an Intel Core i7 CPU/3.60 GHz with win10 operating system.

Experiment Settings

The proposed method was evaluated on cases generating 50 customers and 100 customers with 3 depots separately, which were denoted by "c503d" and "c1003d." The time window for every customer was randomly generated for the cases "c503d" and "c1003d," respectively from 0 minute to 15 minute and from 0 minute to 30 minute. The early punishment coefficient β_i^e was distributed from 0 to 0.5, and late punishment coefficient β_i^l distributed from 0 to 1 were randomly generated. The maximum capacity of every vehicle is 100 and 300 for the cases, "c503d" and "c1003d," respectively. The speed of every vehicle was set as 60 km/h. The arithmetic cases for the test were generated by referring to the reference (Ho et al., 2018), in which the coordinates of the customers were uniformly distributed on the two-dimensional plane *[0,10km]* × *[0,10km]*, and the demand of the customers were uniformly distributed on *[1,10]*.

There are some parameters that were set in the training phase and test phase for MADRLL. Each vehicle was distributed on a different, original depot. In the model training phase, the training epoch, batches per epoch, the number of instances per batch and the learning rate in Adam optimizer were set to 150, 2500, 512 and 1×10^{-4} respectively for the cases with size c50d3 and c100d3. In the test phase, the number of arithmetic cases were set to 10000 under the corresponding distributions of the c50d3 and c100d3 problems. The performance indicators of the model were adopted with average path length and average solving time. The shorter the average path length is; the better the strategy is. Shorter time of average solution indicates a higher efficiency of the model.

Validation of the Multi-Agent Model

Different from single-agent reinforcement learning, MADRLL contains multiple agents, which not only need to have accurate knowledge of the current environment, but also need to integrate the feature information of the other agents to achieve the cooperation effect. The interaction among multiple

agents in MADRLL is implemented through feature fusion in the embedding layer of the decoder, which is operated according to Eq. (18). The state features of the current agent and the other agents are fused in the current environment in the decoder. Thus, the joint optimal action is considered before the current agent makes a decision to select a next node. In contrast, the current optimal action is considered for the single-agent learning mode since there is feature information of the current agent in the embedding layer of the decoder. To test the validity of the proposed model, experiments on instances of sizes c50d3 and c100d3 with various numbers of agents were conducted. The parameter of attention modules λ was set to 3, and the parameter of attention heads K were set to 8. The process of the training cost curves for MADRLL was visualized in Fig. 3.

Viewing from Fig. 5, MADRLL with multiply agents could converge faster and better in the training process than that one with single-agent learning mode. The best convergence was that the algorithm with three agents on arithmetic cases c50d3 and c100d3 and the convergence speed and results on c50d3 were better than c100d3, which indicated that the algorithm would be better with the scale decrease. The algorithm could obtain obvious advantage when the number of agents is close to the number of depots of instance while the worst performers were the single agent, and this difference becomes more pronounced as the scale of the customer increases.

Performance Comparison With Other Algorithms

To verify the advantages of the proposed algorithm, the compared experiments were carried out on instances c50d3 and c100d3, respectively. The comparison algorithms included a reinforcement learning (RL), Google OR-Tools, a hybrid genetic algorithm (HGA2) and ant colony algorithm (ACO). HGA2 was a genetic algorithm combined with neighborhood search and ISP operator, which initialized population with NNH strategy. In the test, the population size was 30, crossover probability was 0.4, variance probability was 0.2, and the number of iterations is 500 for c50d3 size problem and 1000 for c100d3 size problem. The ACO was used with 50 ants, 10 best ants and 0.95 decay rate of pheromone. The exponent of pheromone was set to 0.9 and the maximum number of generations was set to 1000. The results of the comparison experiment were shown in Fig. 4.

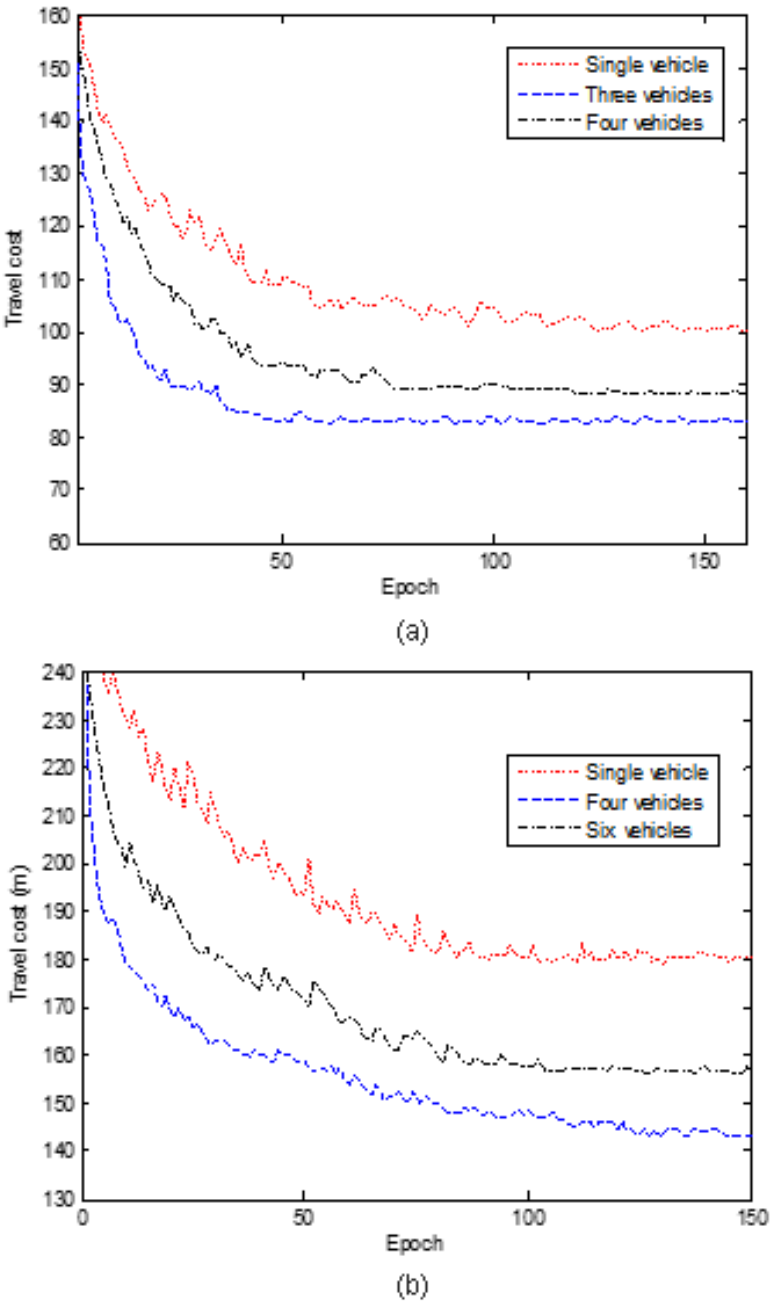
Fig.4 illustrated the mean travel cost and mean execution time for every instance under the algorithms contained HGA2, ACO, OR-Tools, RL and MADRLL. Compared with the HGA2, ACO, OR-Tools and RL, MADRLL could achieve significant improvements in solving quality and computational efficiency in various situations. To generate the suboptimal solutions for 100-customer in ACO was difficult, while superior solutions could be generated by MADRLL with low cost online calculating effort. MADRLL was effective with the changes of conditions, which is a distinct advantage compared with some heuristic algorithms.

Robustness Tests

Generally, it was indispensable to deal with some uncertainties and variations environments for the agency and the high executing time and resources would be required for training the model for every case. It makes sense to research the robustness of MADRLL. After the model of MADRLL was trained, it could be generalized to solve any similar scale problem. To validate the robustness performance of the MADRLL model, different scale models contained varying customers and depots were experimented with MADRLL, RL and OR-Tools. The illustrations of generalization ability of the MDVRPTW comparing to other algorithms were shown in Table 2 and Table 3.

Viewing from Table 2, the superior performance of MADRLL demonstrated effectiveness to the variations in numbers of depots and customers in the travel cost and executing time comparing to OR-Tools and RL. With the changes of number of depots, the model that has been trained could produce optimal solutions. MADRLL model pretrained for the case “c50d3v3” and “c50d3v4” to solve optimal solutions under various conditions, and the results shown the proposed method still outperforms other algorithms.

Figure 3. Performance comparison for various numbers of agents (a) Scale of c50d3, (b) Scale of c100d3



It could be seen from Table 3 that the travel cost and executing time were increasing with the increment of customers for OR-Tools, RL and MADRLL in the case of a fixed number of depot and vehicle. Compared to OR-Tools and RL, MADRLL had the lowest travel cost and executing time. The executing time of MADRLL was relatively close to RL, while it was much faster than OR-Tools. The proposed method was able to quickly solve MDVRPTW under conditions of varying numbers of

Figure 4. Comparison of training result for different algorithms (a) Travel cost, (b) Executing time

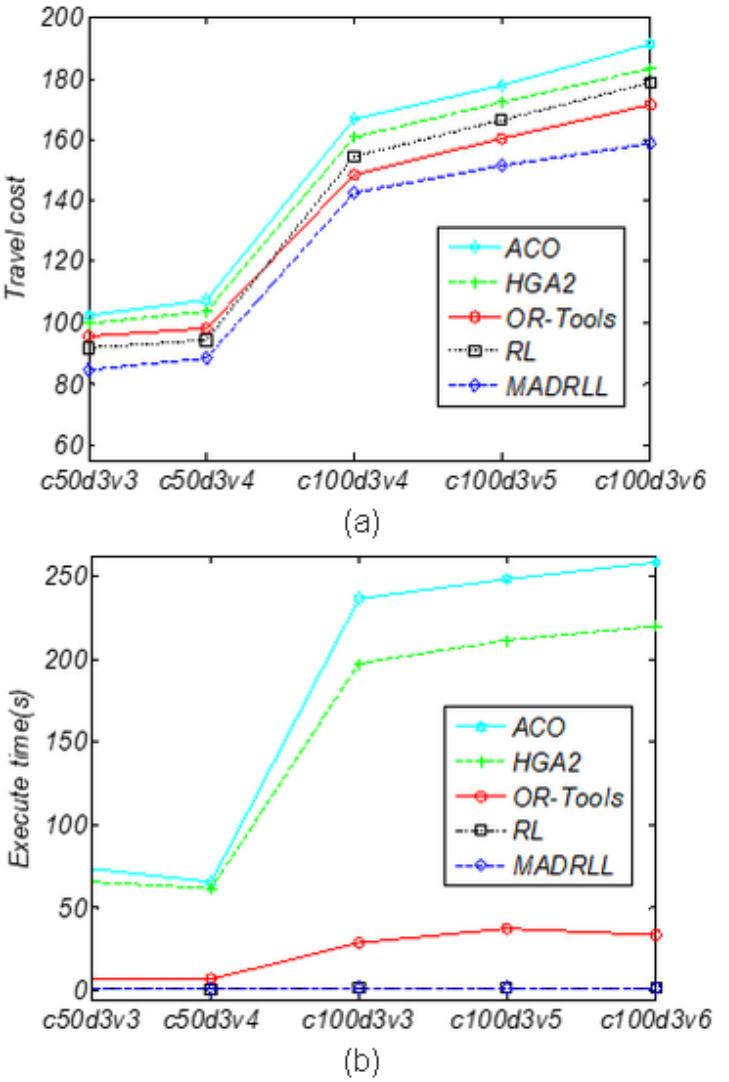


Table 2. Performance comparisons on scenarios with varying number of depots

Case	OR-Tools		RL		MADRLL	
	Cost	Time	Cost	Time	Cost	Time
c50d1v3	99.8163	11.8136	98.6375	1.0473	95.4251	0.8148
c50d2v3	97.4251	9.1255	94.4234	0.9432	89.4232	0.7861
c50d3v3	95.6362	7.4151	91.8182	0.8837	84.3164	0.7557
c50d1v4	101.2947	8.4578	103.6243	0.9114	97.6217	0.7970
c50d2v4	98.7263	7.7857	99.4218	0.8752	93.4242	0.8581
c50d3v4	97.5436	6.7216	94.4239	0.8561	88.4237	0.8253

Table 3. Performance comparisons on scenarios with varying number of customers

Case	OR-Tools		RL		MADRLL	
	Cost	Time	Cost	Time	Cost	Time
c50d3v3	95.63622	7.4151	91.8182	0.8837	84.3164	0.7557
c45d3v3	91.4271	6.5262	86.7217	0.8247	79.7265	0.6642
c40d3v3	85.6375	5.7346	82.5305	0.7363	74.3343	0.5614
c100d3v5	160.3927	37.2907	166.2946	1.7516	152.2917	1.7153
c95d3v5	153.4246	33.4253	160.4233	1.6705	146.4225	1.5921
c90d3v5	146.3203	27.4207	155.4617	1.6028	142.8648	1.5318

customers and depots. It also could efficiently search for good routes even if the scale of customers or depots was increased.

It is significant to explore how the proposed method performs for those vehicles with different capacities. The initial vehicle capacity $\hat{q}_m^{t=0}$ was variable during the experiments. "c50d3v3q160" indicates a case with 50 customers, 3 depots, and 3 vehicles with a capacity of 160. Table 4 recorded the performance comparison on scenarios with uncertain vehicle capacity for algorithm OR-Tools, RL and MADRLL, which showed that the proposed model had better generalization and scalability compared to OR-Tools and RL.

Table 4. Performance comparisons on scenarios with uncertain vehicle capacity

Case	OR-Tools		RL		MADRLL	
	Cost	Time	Cost	Time	Cost	Time
c50d3v3q100	102.4715	8.3286	98.2745	0.9252	91.2753	0.7904
c50d3v3q130	95.63622	7.4151	91.8182	0.8837	84.3164	0.7557
c50d3v3q160	98.5382	9.2972	94.5834	0.8558	87.5387	0.7416
c100d3v5q240	171.0586	28.4355	172.7647	1.5745	158.6468	1.5185
c100d3v5q270	160.3927	37.2907	166.2946	1.7516	152.2917	1.7153
c100d3v5q300	153.2479	27.0713	158.7451	2.1826	149.0688	1.8826

Table 5. Performance comparisons on scenarios with uncertain logistics delivery time

Case	OR-Tools		RL		MADRLL	
	Cost	Time	Cost	Time	Cost	Time
c50d3v3	95.63622	7.4151	91.8182	0.8837	84.3164	0.7557
c50d3v3±25%	99.2661	6.9133	94.1097	0.9075	91.3560	0.78624
c50d3v3±50%	108.7037	7.7255	102.5541	0.9467	98.1749	0.81527
c100d3v5	160.3927	37.2907	166.2946	1.7516	152.2917	1.7153
c100d3v5±25%	164.4665	36.2766	171.1625	1.9391	153.1453	1.8079
c100d3v5±50%	173.4963	37.8125	183.6756	2.4755	156.5741	1.9133

Due to possible traffic conditions during logistics delivery, such as quality of the route, driver fatigue, severe weather, seasons, and multiple depot operations during weather disruptions, the preset values in the experiment may deviate. To verify the robustness of the proposed method, the fluctuation of logistics delivery time with ±25% or ±50% were considered. Table 5 showed performance comparisons on scenarios with uncertain logistics delivery time for OR-Tools, RL and MADRLL. It can be seen that the performance of MADRLL is better than the other algorithms, although they could deal with different logistics delivery time fluctuations well. It indicated that the proposed method has good generalization and scalability.

CONCLUSION

This paper presents a multi-agent deep reinforcement learning with local search strategy to solve the MDVRPTW. The multi-head attention was used to extract important features from observations in raw high-dimensional environment within time window network in encoder. A decoder architecture with multiple agents was constructed to generate a policy to visit the customer at every step successively. MADRLL could perform vehicle path planning for MDVRPTW on global optimizing by learning mutual cooperative actions of multi-agent. By combining the local search strategy to improve the quality of solutions, the speed and quality of offline training and search test by the proposed algorithm could achieve good results. The experiments showed that the MADRLL could obtain superior performance in terms of solution quality and robustness under varying numbers of depots and customers comparing with the other algorithms. The next work will focus on a wider range of optimal problems by extending the reinforcement learning model to facilitate real-world applications. There is a need to explore the potential of learning-based methods for more complex vehicle routing problems.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

FUNDING STATEMENT

This research was supported by the Characteristic Innovative Projects of Guangdong Ordinary Colleges and Universities under Grant No. 2023KTSCX351. Higher education specialization Project for Guangdong Education Science Planning under Grant No. 2023GXJK946 and No. 2023GXJK947, University Quality engineering and Teaching Reform Project under Grant No. swjy23-004 and No. swjy23-002. School Scientific Foundation under Grander No. SKQD2021B-035 and No. SKQD2021Y-031. Higher education collaborative education Foundation of the Education Ministry under Grander No. 202101347006 and No. 202101355031. Key Research Platform for Ordinary Universities of the Education Department of the Guangdong Province under Grant No. 2022CJPT023.

AUTHOR CONTRIBUTIONS

Conceptualization, F.Y. and M.C.; algorithm design, F.Y.; mathematics modeling, F.Y. and D.Z.; experiment and analysis, X.Y. and M.C.; writing—original draft, D.Z., writing—review and editing, X.Y., F.Y. and M.C. All authors have read and agreed to the published version of the manuscript.

AUTHOR NOTE

Fahong Yu: https://orcid.org/0000-0002-9342-6419.

Correspondence concerning this article should be addressed to Fahong Yu, Center of Intelligent Computing and Security Research, Shanwei Institute of Technology, Guandong, 516600, China. Email: fhyu520@whu.edu.cn; Meijia Chen, Shanwei Institute of Technology, Guandong, 516600, China; Xiaoyun Xia, Jiaxing university, Zhejiang, 430010, China.

REFERENCES

Andelmin, J., & Bartolini, E. (2022). An exact algorithm for the green vehicle routing problem. *Transplantation Science*, *51*(4), 128–303.

Asaf L., Uri Y., (2013). Nonoblivious 2-Opt heuristics for the traveling salesman problem, *International journal Journal of Networks, 62* (3), 201-219.

Bono, G., Dibangoye, J. S., Simonin, O., Matignon, L., & Pereyron, F. (2023). Solving multi-agent routing problems using deep attention mechanisms. *IEEE Transactions on Intelligent Transportation Systems*, *22*(12), 7804–7813. doi:10.1109/TITS.2020.3009289

Braekers, K., & Nieuwenhuyse, K. I. (2020). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, *99*(7), 300–313.

Gu, Z., Zhu, Y., Wang, Y., Du, X., Guizani, M., & Tian, Z. (2023). Tian, Applying artificial bee colony algorithm to the multidepot vehicle routing problem. *Software, Practice & Experience*, *52*(3), 756–771. doi:10.1002/spe.2838

Hiermann, G., Puchinger, J., Ropke, S., & Hartl, R. F. (2022). The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, *252*(3), 995–1018. doi:10.1016/j.ejor.2016.01.038

Ho, W., Ho, G., Ji, P., & Lau, H. C. W. (2018). A hybrid genetic algorithm for the multi-depot vehicle routing problem [J]. *Engineering Applications of Artificial Intelligence*, *21*(4), 548–557. doi:10.1016/j.engappai.2007.06.001

James, J., Yu, W., & Gu, J. (2019). Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *20*(10), 306–317.

Kai, A., Marc, P., Miles, B., & Anil, A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. doi:10.1109/MSP.2017.2743240

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization[C]. *The 3rd International Conference on Learning Representations*. San Diego, 2015: 1-11.

Li, J., Ma, Y., Gao, R., Cao, Z., Lim, A., & Song, W. (2021). Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, *52*(12), 572–585. PMID:34554923

Li, Y., Soleimani, H., & Zohal, M. (2023). An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives. *Journal of Cleaner Production*, *22*(7), 1161–1172.

Lin, B., Ghaddar, B., & Nathwani, J. (2021). Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, *23*(8), 28–38.

Luis, G., Markus, L., & Mario, R. (2019). Layered graph approaches for combinatorial optimization problems. *Computers & Operations Research*, *102*(7), 22–38.

Macedo, R., Alves, C., Valério, C., Clautiaux, J. M., & Hanafi, S. (2022). Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, *214*(3), 536–545. doi:10.1016/j.ejor.2011.04.037

Pan, W., & Liu, S. (2023). Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Applied Intelligence*, *53*(1), 405–412. doi:10.1007/s10489-022-03456-w

Ren, L., Fan, X., Cui, J., Shen, Z., Lv, Y., & Xiong, G. (2022). A multi-agent reinforcement learning method with route recorders for vehicle routing in supply chain management. *IEEE Transactions on Intelligent Transportation Systems*, *23*(9), 410–420. doi:10.1109/TITS.2022.3150151

Silva, M. L., Souza, S. D., Souza, M. F., & Bazzan, A. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, *131*(10), 148–171. doi:10.1016/j.eswa.2019.04.056

Stodola, P., & Nohel, J. (2023). Adaptive ant colony optimization with node clustering for the multi-depot vehicle routing problem. *IEEE Transactions on Evolutionary Computation*, *22*(10), 277–291.

Wang, Y., Li, Q., & Guan, X. (2023). Collaborative multi-depot pickup and delivery vehicle routing problem with split loads and time windows. *Knowledge-Based Systems*, *231*(4), 174–189.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256. doi:10.1007/BF00992696

Zhang, K., Li, M., Zhang, Z., Lin, X., & He, F. (2020). Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C, Emerging Technologies*, *121*(11), 216–231. doi:10.1016/j.trc.2020.102861