Applied Intelligence (2023) 53:25881–25896
https://doi.org/10.1007/s10489-023-04881-1

Routing optimization with Monte Carlo Tree Search-based multi-agent reinforcement learning

Qi Wang[1] · Yongsheng Hao[2,3]

Accepted: 10 July 2023 / Published online: 14 August 2023

Abstract
Vehicle routing (VRP) and traveling salesman problems (TSP) are classical and interesting NP-hard routing combinatorial optimization (CO) with practical significance. While moving forward with artificial intelligence, researchers are paying more and more attention to applying machine learning to classical CO problems. However, traditional reinforcement learning faces challenges like reward sparsity and unstable training, so it is necessary to assist agents in finding high-quality routings in the initial model training stage to obtain more positive feedback. This paper proposes a novel Monte Carlo Tree Search (MCTS)-based two-stage multi-agent reinforcement learning training pipeline (MCRL) in which we also design a multifunctional reward function, improving efficiency, accuracy, and diversity to guide agents to learn the routings over graphs better. Besides, previous approaches are frequently too sluggish in runtime to be useful in contexts with sparsely connected networks and uncertain traffic. As an alternative, we design a model based on graph neural networks that can execute multi-agent routing in a sparsely connected graph with constantly changing traffic circumstances. Also, the agents are better equipped to collaborate online and adjust to changes thanks to our learned communication module.

Keywords NP-hard problems · Combinatorial optimization · Graph neural networks · Reinforcement learning · Monte Carlo Tree Search

1 Introduction

Combinatorial optimization (CO) problems [1] (Fig. 1 illustrates an example of VRP) usually include NP-hard and P-problems, and fast solving them is of central theoretical significance and practical application value. Traditional approaches primarily design corresponding approximation or heuristic algorithms tailored to specific problems (e.g., ant colony, genetic, simulated annealing, etc.) [2–5], but they cannot effectively use previous experience for different instances of similar problems. There are inherent similarities between problems occurring in the same application area [6], but traditional approaches do not systematically take advantage of this.

Therefore, people hope to find a general way to address optimization problems, dig out the essential information on the problems with offline learning and improve the efficiency and quality of solving problems by automatically updating the solution policy online. Deepmind has revolutionized artificial intelligence by showing that deep learning and reinforcement learning [7] can solve some CO problems since playing Go [8], resource allocation [9], and matrix decomposition [10] are about searching for solutions across large combinatorial spaces. We can also establish an appropriate mathematical model for CO problems, apply deep neural networks for feature representation, and use appropriate search strategies to reduce the solution space [11]. We can gradually accumulate experience to guide the solutions of future (unknown) instances. For example, some recent works adopt the encoder-decoder paradigm based on the attention mechanism [12] to deal with the node sequences in CO problems [13–15]. Some works apply graph neural

✉ Qi Wang
qiwang@dlmu.edu.cn

✉ Yongsheng Hao
hyslove@163.com

1 Information Science and Technology College, Dalian Maritime University, Dalian 116026, China

2 School of Mathematics and Statistics, Nanjing University of Information Science & Technology, Nanjing 210044, China

3 Network Center, Nanjing University of Information Science & Technology, Nanjing 210044, China

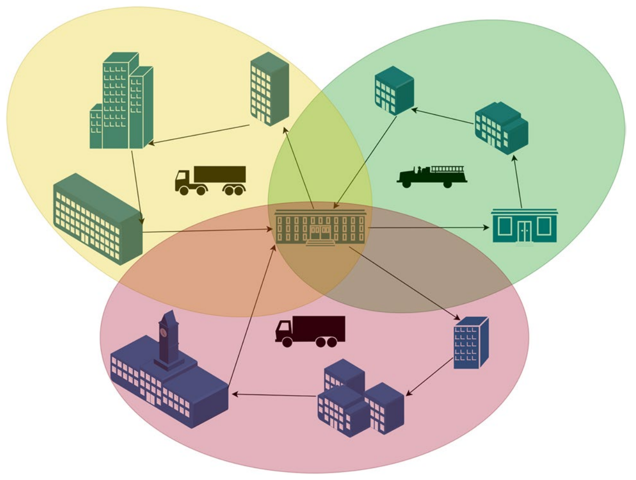


Fig. 1 A simplified diagram of the relative location of customer points and a service center point, where vehicles (their cargo capacity) may differ, with different requirements for each customer point

networks (GNNs) [16] to aggregate the information on nodes and edges to learn the topological structure of CO graph instances [17–19]. Then they apply reinforcement learning with a search strategy to optimize the model's parameters and output solutions.

Deep reinforcement learning regards general graph routing optimization problems as data points under a given distribution. After learning the probability distribution of data points, it can be generalized to other problem instances with a similar distribution. Traditional heuristic algorithms based on expert experience tend to fall into local optimality without realizing it, but deep reinforcement learning can break through this limitation. Solving problems one-to-many in an automated manner is attractive and can bring substantial economic benefits and eliminate inefficient manual work, which may be one of the core goals of artificial intelligence. Despite showing promising results, existing learning-based works have several significant limitations. For example, in modeling, the number of traditional GNN parameters increases with the graph instances size [20], which may lead to being too heavy to be trained effectively. Recurrent neural network (RNN)-based approaches [14, 15, 21] may also affect parallel computation as the size of graphs increases. Reinforcement learning improves the model's generalizability, but its accuracy is currently inferior to supervised learning [22]. More importantly, reinforcement learning is notoriously unstable, and it often encounters the problem of sparse rewards in the initial training phase.

Moreover, the generated solutions are not diverse enough, as existing methods can only train one construction policy and apply sampling or beam search to create solutions from the same policy. The only source of diversity is a relatively deterministic probability distribution, which is far from sufficient. Previous methods' multi-agent settings are rarely explored and frequently evaluated on rudimentary planar graph benchmarks. Moreover, none of these approaches were intended for dynamic settings where online communication might be quite valuable. This is a challenging issue with several fleet management applications to accomplish the same objective, including ride-sharing and robot swarm mapping.

We intend to address these challenges by contributing to modeling, training, and coordinating the routing of multiple agents. Specifically, we first design an attentional policy network by combining a message-passing neural network and a transformer's encoder. Then, we increase the diversity of the generated solutions by integrating multi-agent communication and multiple decoders. Finally, we designed a training pipeline to train the policy network using supervised learning followed by reinforcement learning.

In contrast to previous pure reinforcement learning-based approaches, we started training a supervised learning policy that provides fast and efficient learning updates, high-quality gradients, and immediate feedback throughout the training process. CO has many similarities with the game of go (GO for short), such as exploring a vast solution space and a clear objective function and constraints to evaluate the current policy. AlphaGo series [23] has proven the MCTS sufficient for large-scale combinatorial space, so it should also be a reasonable choice to introduce MCTS into CO. So, we apply reinforcement learning with the MCTS to improve the supervised policy network, which will adjust the policy to find the best solution.

To summarize, the contributions of this paper are threefold:

- We propose the Multi-Agent GNN policy network, a distributed deep neural net, to coordinate a swarm of moving objects toward a predetermined objective. Specifically, each agent engages in local planning within a learned GNN that uses inter-agent communication using a cutting-edge learned communication protocol that employs an attention mechanism.
- We propose a training pipeline suitable for routing optimization, effectively integrating supervised learning, reinforcement learning, and MCTS, and improved algorithm accuracy and training stability. In addition, we use multiple decoders for multiple agents to further enhance the diversity of policies.
- We propose a precise reward function for routing optimization, combining global, length, and efficiency. In addition, we use the classic A* algorithm to fine-tune the reward function and MCTS further to improve the power and accuracy of the search algorithm.

2 Related work

More and more researchers are applying machine learning, especially deep learning, and reinforcement learning, to CO to solve sequential decision problems in graphs by combining the perceptual ability of deep learning and the inference ability of reinforcement learning. According to the differences in deep learning modeling, we broadly classify learning-based construction heuristics into attention-based and GNN-based. Next, we present some representative methods in recent years. Bengio et al. [1] and Wang et al. [12] extensively surveyed the application of deep reinforcement learning in CO.

Recent deep learning models such as pointer networks [13], transformers [12], and others based on RNN and attention are gradually applied to routing optimization problems. Bello et al. [21] pioneered using reinforcement learning (an actor-critic algorithm) to train a pointer network in an unsupervised manner, taking each instance as a training sample and using routing length to make unbiased Monte Carlo estimates of the policy gradient. Nazari et al. [14] improved the pointer network and combined it with the actor-critic algorithm to design an end-to-end framework for solving VRP beyond classical heuristics and Google's OR-Tools for medium-sized instances. Kool et al. [15] designed a model based on an encoder-decoder structure using the transformer, which first applies an encoder to obtain node context information via node embedding and message passing using the graph as input. Then they apply the REINFORCE with rollout baseline to greedily decode the node sequence. Following Kool et al., Xin et al. [24] proposed a multi-decoder attention model to train various policies. Compared with existing methods that only train one policy, it increases the finding of reasonable solutions.

GNNs [16] have been powerful tools used to process graph data in recent years, and they can effectively aggregate and learn the structure information of graphs. Dai et al. [13] applied structure2vec to embed graph instances, models that reflect combinatorial structures better than those based on sequence-to-sequence. Then they used the DQN algorithm [6] to construct feasible solutions by continuously adding nodes and maintaining feasible solutions to meet the graph constraints of problems [20]. Some works [25, 26] combine the advantages of supervised learning and reinforcement learning. Still, they are two-stage or completely independent learning processes, whereas our training pipeline "seamlessly" links supervised learning and reinforcement learning using shared parameters of the Monte Carlo policy gradient.

In addition, more and more works focused on iteratively improving the quality of solutions by learning improvement heuristics or exact algorithms in solution solvers. Chen et al. proposed NeuRewriter [25] to learn a policy to select heuristics and rewrite local components of the current solution to improve it until convergence iteratively. They divided the policy into region and rule selection components, applied a neural network to parameterize each component, and trained the neural network using the actor-critic algorithm. The L2I [26] proposed by Lu et al. has two parts, namely the improvement controller (operators) and the perturbation controller (operators), which complement each other to update and iterate the initial solution. Given a random initial solution, L2I learns to iteratively finalize with an improvement operator selected by the controller based on reinforcement learning. Zheng et al. [27] proposed a variable policy reinforcement learning method that combines three types of reinforcement learning (Q-learning, Sarsa, and Monte Carlo) and the famous Lin-Kernighan-Helsgaun (LKH) algorithm. Delarue et al. [28] developed a deep reinforcement learning framework with a value function. It has a combinatorial action space in which the action selection problem is clearly expressed as a mixed-integer optimization problem. Cappart et al. [29] proposed a general hybrid method based on deep reinforcement learning and constraint programming for combinatorial optimization. The core is the dynamic programming formulation, which serves as a bridge between these two technologies. Similar methods include [30–36], etc. Although these methods improve solution quality, they rely too much on known initial solutions or take too long to train iteratively.

Compared with previous reinforcement learning methods, our method is different in the following aspects: (1) We use multiple agents to simulate multiple vehicles and design a communication mechanism to enable multiple agents to transmit information to each other, which enables each agent to have a global vision and is more conducive to the exploration on the graph instance (environment). (2) The design of the reward function is vital for reinforcement learning because it directly affects learning efficiency and effectiveness. Therefore, we tailor a multi-agent reward function that balances global efficiency and accuracy. (3) We design a novel graph neural network based on a message-passing network for multi-agent communication to effectively process the dynamic node sequence, which aligns more with real-life routing optimization. (4) We design a training pipeline combining supervised learning and reinforcement learning to stabilize training and improve accuracy and introduce the A* algorithm into MCTS so that it can further fine-tune the reinforcement learning and search algorithm together with the reward function, which can make the reward function control agents more effectively.

3 Methodology

3.1 Problem definition

VRP (as illustrated in Fig. 1) means vehicles can drive orderly through appropriate routings to minimize the total cost under certain constraints [37]. Usually, the optimal route for multiple vehicles is the one that minimizes the total distance traveled. Assume that the optimal solution is equivalent to assigning only one vehicle to access all nodes and finding the shortest path for that vehicle. In this case, it becomes the traveling salesman problem (TSP).

We first give the mathematical model of TSP, specifically:

$$min \sum\nolimits_{i,j \in I} d_{ij} x_{ij} \tag{1}$$

$$\sum\nolimits_{i \in I, i \neq j} x_{ij} = 1, \forall j \in I \tag{2}$$

$$\sum\nolimits_{j \in I, j \neq i} x_{ij} = 1, \forall i \in I \tag{3}$$

$$\sum\nolimits_{i \in S} \sum\nolimits_{j \in S, j \neq i} x_{ij} \leq |S| - 1, \forall S \subset I, 2 \leq |S| \leq n - 1 \tag{4}$$

$$x_{ij} \in \{0, 1\} \tag{5}$$

where (1) represents the objective function, the shortest distance. d_{ij} represents the distance between node v_i and node v_j; x_{ij} represents the decision variable (5), and when its value is 1, it means that the node v_j and node v_j are adjacent in the path—constraints (2) and (3) guarantee that each node is visited only once. Constraint (4) guarantees that there will be no subrings in the tour (if S nodes form a loop, then at least S edges are needed, thus avoiding the generation of subrings). I represents the collection of all nodes. Note that the symbols in the above definition are not directly related to the following.

We then represent an instance of VRP by an undirected weight graph $G(V, E, \prod)$, where V represents the set of all nodes; we employ v_i to represent each node in the graph, and $v_i \in V$, E represents the set of all edges in the graph, and $e_{ij} = (v_i, v_j) \in E$, $\prod$ is the adjacency matrix.

3.2 Markov decision process

To apply reinforcement learning to VRP and TSP, we need to model these two problems as the Markov decision process (MDP). We define a deterministic MDP as (S, A, P, R), where S is the state space, namely the set of all states. A refers to the space of the actions performed by the agent, which come from the state $s \in S, P : S \times A \to S$ refers to the deterministic state transfer function (the state changes after the execution of the action). R is the immediate reward function.

Actions Starting from the start node v_0, the agent follows the policy network to take an action that selects the next node v_i as part of a promising path $(v_0, v_1, \ldots, v_0)$, to expand its paths at each step to guide it back to v_0. Then, it set out to find the paths that have not been traveled until all nodes in the graph have been traversed.

States We define the part of the tours that the agent finds as S, and a set of termination states as S_{end}. Given a state s, the agent repeatedly selects actions from A and moves to the next state until it stops when $s \in S_{end}$.

Transition When an action (v_i) is added to part tours, the state changes from s to $\acute{s}$, and $P(s, a, \acute{s}) = 1$.

Rewards The immediate reward for the agent at each time step is r_t. The path length is the reward for a general routing problem, but several factors affect the path quality the agent finds. To better apply the reward function to control the agent to find the optimal routing (path), we divide the total reward function (r_{total}) into the following parts:

Global: From a global perspective, if the agent performs a series of actions from the central node and then returns to the central node, it will be considered successful:

$$r_G = \begin{cases} +1, & \textit{if all nodes are traversed} \\ -1, & \textit{otherwise} \end{cases} \tag{6}$$

Length: (For TSP) When all nodes in the graph G are traversed, the ordered sequence $\widehat{S} = \{v_0, v_1, \ldots, v_n\}$ can be calculated by weights on the edges:

$$L_v = \sum\nolimits_{i=0}^{|\widehat{S}|-1} w_{i,i+1} + w_{|\widehat{S}|,0} \tag{7}$$

$$r_L = {}^{1}\!\big/_{L_v} \tag{8}$$

(For VRP) We can choose to apply the subgraph sampling algorithm [38] to divide the graph into subgraphs that all contain the central node (v_0). At this point, the problem is transformed into solving TSP on each subgraph.

When the node v is added to S, the length of the partial sequence $\widetilde{S} = S \cap v$ can be calculated by the following formula:

$$l_v = \sum\nolimits_{i=0}^{|\widetilde{S}|-1} w_{i,i+1} \tag{9}$$

Efficiency: From a local perspective, we also hope that the agent can choose a short path each time, and a shorter path can improve the efficiency of reasoning by limiting the interaction length between reinforcement learning and the environment, specifically:

$$r_E = {}^{1}\!\big/_{(L_v - l_v)} \tag{10}$$

Multi-Agent To learn more diversified policies, we have introduced multi-agents [39, 40] to represent vehicles separately and assume that each agent can broadcast to other agents in the fleet to deliver messages. For example, one agent v_a^i of K agents $\{v_a\}_{i=1}^{K}$ takes action a_t^i at time step t, and its state changes from s_t^i to s_{t+1}^i, and its policy includes communication messages $\{c_t^j\}$ sent by other agents.

3.3 Attentional policy network

In principle, we can apply any GNNs to parameterize the policy function $\pi_\theta(a|s)$), which maps the state vector s to the probability distribution of all possible actions. Still, traditional GNNs are often limited in computational efficiency and are difficult to extend to large-scale graphs. Therefore, we propose a novel GNN, graph message passing pointer network (GMPPN) (as illustrated in Fig. 2), specifically for dynamic sequence decision tasks on graphs. The encoder generates a representation of all input nodes in graph G, and the decoder selects a routing sequence among the input nodes through pointers, where the constraints are realized by masking.

Encoder We first combine the message-passing neural network (MPNN) [16] and the encoder in the transformer [12] to design our encoder. Through the feature x_i of the node in the graph, we apply linear transformation to get its hidden feature:

$$h_i^{(0)} = W_1 x_1 + b_1 \tag{11}$$

$W_i(i = 1, 2, 3, 4, 5, 6, 7)$ and $b_i(i = 1, 2, 3, 4)$ represent the parameters of the corresponding dimension. We update the node embedding through the self-attention layer. Each self-attention layer comprises two sub-layers: a multi-head self-attention layer and a feedforward layer. After processing, we can obtain the hidden features of the current layer $H^{(l)} = (h_1^{(l)}, h_2^{(l)}, \dots, h_n^{(l)})$. We obtain the node embedding $E = W_2 H$ by linear transformation and obtain the self-attention score by the following equation:

$$SelfAttention(E) = softmax\left(\frac{EE^T}{\sqrt{d}}\right)E \tag{12}$$

We stitch the results after M self-attention (multi-head attention) and then pass through the fully connected feedforward layer to get the final node embedding of the multi-head attention mechanism, specifically:

$$\hat{H}^{(l)} = [head_1; \dots ;head_M]W_3 \tag{13}$$

where $head_i$ represents the result of the $i - th$ self-attention:

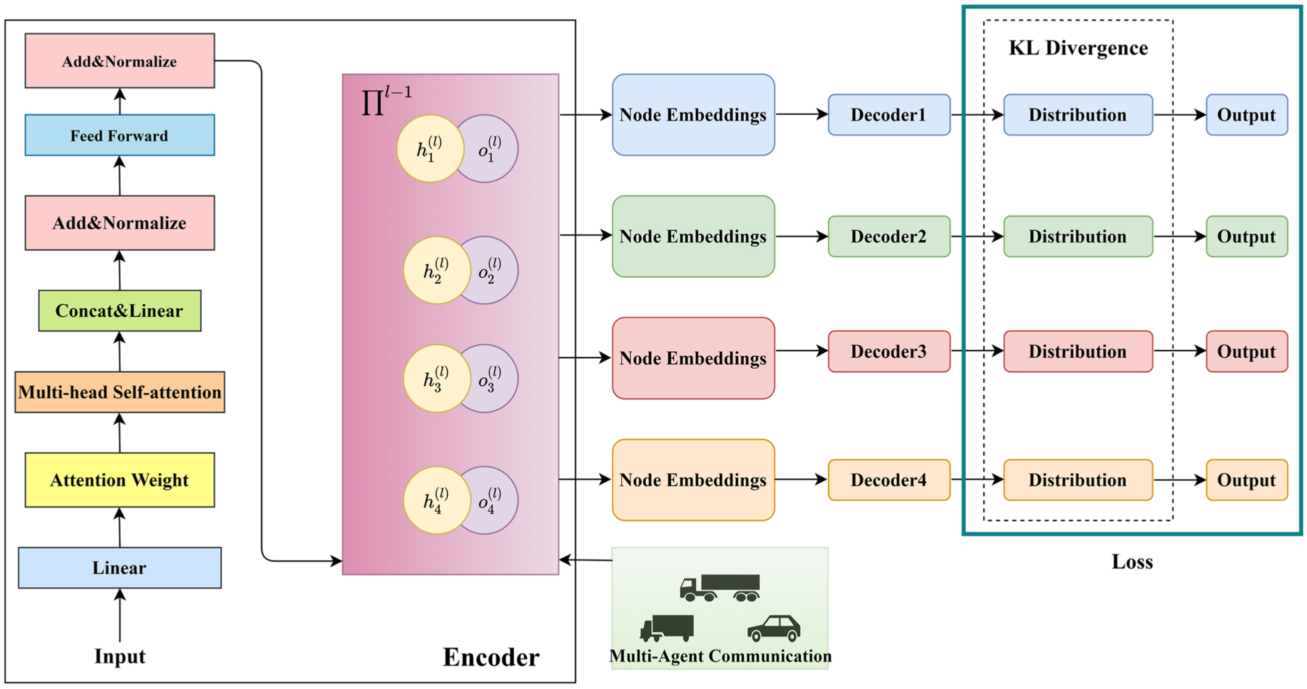


Fig. 2 GMPPN's demonstration includes an encoder, communication models, and multiple decoders (take four decoders as an example)

$$head_i = SelfAttention(W_4 H^{(l-1)}) \tag{14}$$

where $H^{(l-1)}$ is the output of the previous encoder layer. The feedforward layer sublayer is composed of two linear transformations with a rectified linear unit (ReLU) activation:

$$h_i^{(l)} = W_6 ReLU\left(W_5 \widehat{h_i^{(l)}} + b_2 \right) + b_3 \tag{15}$$

In addition to the general GNN functionality, we also want to give GMPNN the ability to handle dynamic node sequence problems. Specifically, for n nodes, at different time steps (or layers), the input is a sequence $O^{(1)}, O^{(2)}, \dots, O^{(l)}$, where $O^{(l)} = (o_1^{(l)}, o_2^{(l)}, \dots, o_n^{(l)})$, $o_i^{(l)}$ represents operations on the node i in layer l (adding or deleting edges). Our task is to predict the vector $y^{(l)}$ (readout function) through the sequence $O^{(1)}, O^{(2)}, \dots, O^{(l)}$. For each propagation, we combine the operation $o_i^{(l)}$ of the current layer with the hidden feature $h_i^{(l-1)}$ of the upper layer to obtain the latent space vector $z_i^{(l)}$:

$$z_i^{(l)} = ReLU\left(W_7 \left[o_i^{(l)}; h_i^{(l-1)} \right] + b_4 \right) \tag{16}$$

Here we apply a fully connected neural network with nonlinear activation ReLU, and W_2, b_2 are parameters. At this point, we get the initial representation of the current layer $Z^{(l)} = \left(z_1^{(l)}, z_2^{(l)}, \dots, z_n^{(l)} \right)$, then we apply the adjacency matrix $\prod^{(l-1)}$ and $Z^{(l)}$ to update $H^{(l)}$:

$$H^{(l)} = f\left(Z^{(l)}, \prod^{(l-1)} \right) \tag{17}$$

Here we integrate the adjacency matrix $\prod^{(l-1)}$ of the previous layer to learn the association information between nodes. We introduce MPNN to set the mapping f, that is, the original node information and the edge information obtained by $\prod^{(l-1)}$, specifically:

$$h_i^{(l)} = \widehat{U}(z_i^{(l)}, \max_{\prod_{ji}^{(i-1)}=1} \widehat{M}(z_i^{(l)}, z_j^{(l)})) \tag{18}$$

Here $\widehat{M}$ is message functions, and $\widehat{U}$ is vertex update functions.

Multi-decoder We have learned more diversified policies with multi-agents, so we apply multi-decoders to generate diversified solutions accordingly. We use M to represent the number of decoders (corresponds to the number of agents) with the same structure and m to index each decoder. The model selects the next node visit probability according to an attention-pointing mechanism at each step. Following Xin et al. [24], the formal definition of decoders indexed by m is as follows:

$$h_c = Concat(\overline{h}, \widehat{h}, h_{t-1}) \tag{19}$$

$$g_c^m = Multihead(W_{g^Q}^m h_c, W_{g^K}^m h, W_{g^V}^m h) \tag{20}$$

$$q^m, k_i^m = W_Q^m g_c^m, W_K^m h_i \tag{21}$$

$$u_i^m = Dtanh((q^m)^T k_i^m / \sqrt{d}) \tag{22}$$

$$p_m\left(y_t | X, y_1, \dots, y_{t-1} \right) = softmax(u^m) \tag{23}$$

where h_c is the context embedding, $\overline{h}$ is the mean of the node embedding, $\widehat{h}$ is the starting node embedding, and h_{t-1} is the current node's embedding. We also adopt maximizing the Kullback-Leible (KL) divergence between the output probability distributions of multiple decoders as a regularization in the training process:

$$D_{KL} = \sum_s \sum_{i=1}^M \sum_{j=1}^M \sum_y p^i(y|X,s) log \frac{p^i(y|X,s)}{p^j(y|X,s)} \tag{24}$$

Here $p^i(y|X,s)$ denotes the probability decoder i selects the node y in the state s for the instance X.

Communication For reinforcement learning, it is beneficial for agents to communicate their expected trajectories, thus encouraging more cooperative behavior. We propose a communication model based on attention, which can only realize dynamic communication between agents when necessary.

We use F_i^a to represent the feature of each agent, which are guided by basic domain knowledge (such as agent type or location). We define the communication encoding function $E^c()$, which is applied to all agent features to generate encoding e_i^c and attention vectors $\vec{a}_i$. $E^c()$ is implemented by fully connected neural networks (FCNs):

$$E^c\left(F_i^a \right) = (e_i^c, \vec{a}_i) \tag{25}$$

For each agent, we calculate the pooled feature P_i^f, which is the interaction vector from other agents, weighted by attention:

$$P_i^f = \sum_j e_j^c * Softmax(-\| \vec{a}_i - \vec{a}_j \|^2) * \lambda_{j=i} \tag{26}$$

where $\lambda_{j=i} = 0$ means self-interaction, and $Softmax(-\|a_i - a_j\|^2)$ gives a measure of the interaction among agents. The pooled-feature P_i^f is connected with the original feature F_i^a to form an intermediate feature C_i:

$$C_i = (P_i^f, e_i^c) \tag{27}$$

Here $C = \{c_1, c_2, \dots, c_K\}$, and K is the number of agents. Through linear transformation, we can transform the communication vector into a query Q_c, key K_c, and value vector V_c, and then calculate the aggregated communication as C_i^a (as part of the node feature input):

$$C_i^a = \sum_j softmax(Q_c K_c) \cdot V_c \tag{28}$$

3.4 Training pipeline

For a typical VRP, the reinforcement learning agent will face hundreds or thousands of possible states or actions. Reinforcement learning often suffers from severe reward sparsity problems in the initial stages of training. Inspired by the imitation learning pipeline used by AlphaGo [23], we design a training pipeline (illustrated in Fig. 3) that starts training from a supervised policy to alleviate this challenge. It is hard for the direct training agent to select actions from the initial action space, so AlphaGo first applied the actions of the experts to train a supervised policy network.

AlphaGo's Core Technology AlphaGo trains the neural network with a "pipeline" of two machine learning stages. First, it trains the supervised learning policy network directly from expert actions, which provides fast and efficient learning updates with immediate feedback and high-quality gradients and sample actions quickly during the first presentation (rollouts). Then, it trains a reinforcement learning policy network to improve the supervised learning policy network by optimizing the outcome of the self-play game. This adjusts the strategy towards the correct goal of winning the game rather than maximizing prediction accuracy. Finally, AlphaGo trains a value network that predicts the winner of a reinforcement learning policy network playing against itself. AlphaGo combines the policy and value networks with the MCTS.

Supervised learning In the first training phase, we apply supervised learning to predict the heuristic function, learning the parameters in the policy $\pi(s;\theta)$ obtained using GMPPN. We apply the Monte Carlo policy gradient with supervised policy learning to update the parameters (for agent i) to maximize the expected cumulative rewards.

$$J(\theta) = E_{a\sim\pi_\theta(a^{(i)}|s^{(i)})}\left(\sum_t r_{s_t^{(i)},a_t^{(i)}}\right) = \sum_t \sum_{a\in A} \pi_\theta(a^{(i)}|s_t^{(i)}) r_{s_t^{(i)},a_t^{(i)}} \tag{30}$$

Here $J(\theta)$ is the expected reward for each episode, $+1$ reward is given for each step of a successful episode. We update the approximate gradient of the policy network by inserting the paths found by MCTS:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_t \sum_{a\in A} \pi\left(a^{(i)}|s_t^{(i)};\theta\right) \nabla_\theta log\pi(a^{(i)}|s_t^{(i)};\theta) \\ &\approx \nabla_\theta \sum_t log\pi(a^{(i)} = r_t^{(i)}|s_t^{(i)};\theta) \end{aligned} \tag{31}$$

where $r_t^{(i)}$ belongs to the node sequence.

Reinforcement learning The first stage of the training pipeline is the supervised learning described above, which can be simplified as follows:

$$\nabla\theta \propto \frac{\partial log\pi_\theta(a|s)}{\partial\theta} \tag{32}$$

The second phase of the training pipeline aims to improve the policy network through policy gradient reinforcement learning, which can also be expressed as:

$$\nabla\theta \propto \frac{\partial log\pi_\theta(a_t|s_t)}{\partial\theta} r_t \tag{33}$$

The reinforcement learning policy network and the supervised policy network are identical in structure. Their weights can be initialized to the same value, so we apply the same θ to represent the parameters to be learned.

We designed a path inference algorithm controlled by the reward function (defined in Section 3.2) to retrain the supervised policy network. We treat the two adjacent nodes and their edges as one episode. According to the policy π, agents start from the central node to select nodes according to the policy, which is the probability distribution of all

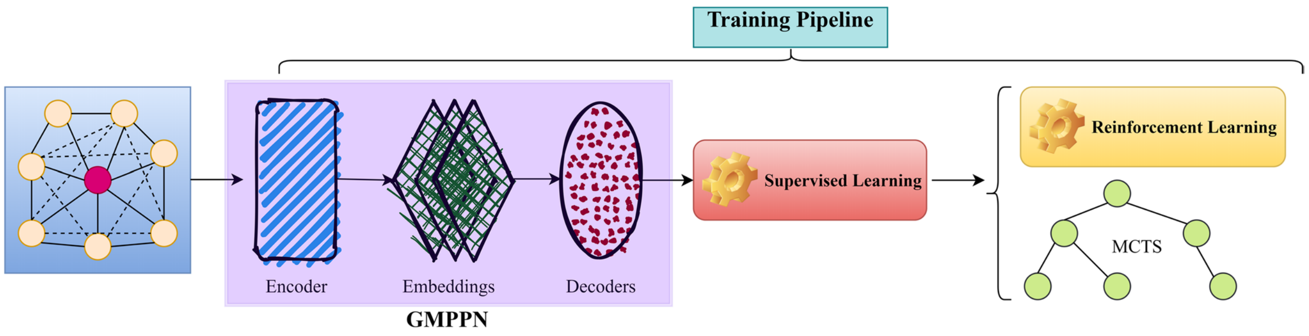


Fig. 3 The complete flow diagram of the training pipeline, in which we apply policy gradient descent for supervised learning after processing GMPPN and then pass the parameters into reinforcement learning and Monte Carlo Tree Search (MCTS) for retraining with reward function

nodes to expand inference paths. Since this action may result in finding a new node or nothing, the failed step will result in the agent receiving a negative reward. Compared with the Q-learning algorithm, the stochastic policy makes the agent not fall into an endless cycle due to repeated wrong steps but keeps the state unchanged. We give an upper limit for the episode to constrain its maximum length (T) to improve training efficiency. If the agent cannot find the target node in T, the episode ends. We employ the following gradient to update the policy network at the end of each episode:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_t log\pi(a^{(i)} = r_t^{(i)}|s_t^{(i)};\theta) r_{total} - k_{KL}\nabla D_{KL} \quad (34)$$

MCTS with WU-UCT MCTS combines the optimal first search tree with the Monte Carlo method, which applies known or trained environmental models to try and play in many environmental models to find better policies. Liu et al. propose a novel parallel MCTS algorithm, WU-UCT (watch the unobserved in UCT) [41], which achieves linear acceleration with little performance loss. The core of the WU-UCT algorithm is to maintain an extra statistic to record how many workers are simulating it on each node and adjust the selection algorithm with it:

$$a_z = argmax_{\widetilde{Q}_i}\left(\widetilde{Q}_i + \beta\sqrt{\frac{2\log(N_z + O_z)}{N_i + O_i}}\right) \quad (35)$$

Here N_z and N_i are the number of accesses to nodes z and i in the tree, respectively, and O_s is the newly added statistics used to calculate the number of rollouts initiated but not completed [41]. To better integrate MCTS into our training pipeline, we introduce the classical A^* algorithm [42] to fine-tune and control policy learning with the reward function.

In formula (15), $\widetilde{Q}_i$ is defined as follows:

$$\varphi(v) = L_v + l_v \quad (36)$$

$$\widetilde{Q}_i = {}^1\!/_{\varphi(v)} \quad (37)$$

where L_v and l_v have been defined in the reward function. l_v is unknown and is the optimal path from the current node v to the target node (see Section 3.2). $\widetilde{Q}_i$ is the best reward under the subtree of node i. We illustrate the detailed process of retraining the policy network using reinforcement learning and MCTS in Algorithm 1.

4 Experiments

In this section, we conduct a series of experiments and perform a detailed experimental analysis. First, we present the dataset (generated data distribution), experimental setup (hyperparameter settings for deep reinforcement learning), and the results of the method proposed in this chapter on a small-scale TSP (including a comparison of results with

Algorithm 1 Retraining with Rewards

```
Input: Restore parameters $\theta$ from the supervised policy;
Initialize fixed-size buffers $B = \{\}$
for $k = 0, 1, ...$ do
    for $episode \leftarrow 1$ to $n$ do
        Initialize state vector for each agent: $s_t \leftarrow s_0$
        Initialize episode length: $steps \leftarrow 0$
        for $t < T$ do
            Perform an MCTS consisting of $S$ simulations
            Extract MCTS-improved policy $\pi(a^{(i)}|s_t^{(i)})$
            Randomly sample action $a^{(i)} \sim \pi(a^{(i)}|s_t^{(i)})$
            Agent $i$ takes an action $a_t^{(i)}$, and observe the next state $s_{t+1}^{(i)}$
        end
            if $r_t = -1$, then Store $(s_t^{(i)}, a^{(i)})$ in $B$
            if $success$ or $t = T$, then $\theta \leftarrow \theta_k$, break
              $steps \leftarrow steps + 1$
          Update $\theta$ by $\nabla_\theta \sum_D log\pi(a^{(i)} = r_t^{(i)}|s_t^{(i)};\theta)(-1)$
          if $success$, then $r_{total} \leftarrow \lambda_1 r_G + \lambda_2 r_L + \lambda_3 r_E$
          Update $\theta$ by Equation (35)
    end
                    $\theta_{k+1} \leftarrow Adam(\theta, \nabla_\theta J(\theta))$
end
```

different methods, ablation studies, learning curves, etc.). Finally, the results of a more complex CVRP (capacity-constrained VRP) are presented (including comparative studies, MCTS analysis for parametric analysis, etc.).

4.1 Data sets and settings

We generate instances with 20, 50, and 100 nodes and use the two-dimensional Euclidean distance to calculate the distance between two cities [15]. We sample the city location coordinates of the two dimensions from a uniform distribution $[0, 1]^2$. For CVRP with 20, 50, and 100 nodes, the vehicle capacity is fixed at 30, 40, and 50, respectively. We sample the demand of each non-warehouse city from the integer $\{1\ldots9\}$.

We embed nodes in the element projection of the 128-dimensional vector. The transformer encoder has three layers of 128-dimensional features and eight heads. Each decoder takes a 128-dimensional vector and eight headers. In the training phase of algorithm 1, we set the batch size to 512, the number of epochs to 100, and the number of training steps to 2000. We set the KL value to 0.01, made the Adam optimizer have 10^{-3} learning rate and 0.96 learning rate delay, with 32 small batches of 50 gradient descent step each iteration. The MCTS performs 300 simulations per move. We store the reward function in a dataset, and MCTS fine-tunes the policy for training. We apply the solution solver LKH3 to get the label data needed for supervised learning in the training pipeline. In our experiments, we only need a tiny amount of label data to drive stable training of the entire framework and get good results, as we will specify later in the experimental task.

We give ample training time to instances of different sizes to gain insight into the data distribution and unlock the model's potential. We have given the maximum training time in which learning usually converges. In terms of implementation conditions, as the number of problem instances increases, the computing resources required also increase, and the (training) effect of the learning method is closely related to the hardware. For example, we can observe that the training speed increases significantly with the number of parallel GPUs. So we ran all our experiments on four parallel NVIDIA GeForce GTX1080Ti GPUs.

We chose some classic heuristics and more recent methods based on learning as our baselines. Given the difficulty of replicating the previous work (such as requiring a large number of computing resources and time), we tried to select experimental tasks and parameter settings similar to those of previous works to facilitate comparison. We extracted the baseline experimental results of the comparison experiment from published papers.

4.2 Effects on small-scale TSP

We first trained and tested MCRL with only one agent on small-scale instances, i.e., TSP20, TSP50, and TSP100 instances, with a training time of 10 min per epoch on TSP20 (with 3 min for supervised learning), 30 min on TSP50 (with 10 min for supervised learning), and TSP100 with a training time of 60 min (of which 20 min for supervised learning). We compare the performance of MCRL on small-scale TSP with previous works and show the results in Fig. 4a, where the approximate ratios of the different methods to the optimal solution are compared (closer to 1 means closer to the optimal solution).

Figure 4a shows that MCRL outperforms traditional algorithms and representative learning-based approaches for small-scale TSP instances. It demonstrates the efficiency and effectiveness of the network architecture and training algorithm proposed in the essential state (with only one agent). Traditional heuristics such as 2-opt, Cheapest, and Closest have significantly lagged behind learning-based methods regarding performance due to their singularity of functionality (2-opt is an improvement heuristic that improves the quality of the solution by swapping the order of the nodes of a given solution. So it results in a higher quality of the solution relative to other traditional construction heuristics), especially since the performance degradation is more evident as the instance size increases. The optimal solutions in the experiments are obtained by a solution solver, which usually integrates multiple exact and heuristic algorithms to enhance solving power and adaptability. Solution solvers can obtain optimal solutions in small-scale instances but, like traditional algorithms, are limited by the problem size and can only solve specific problem instances.

MCRL differs significantly from previous learning-based approaches in that we design a GNN suitable for dynamic sequences on graphs. Multi-agent communication mechanisms are effective when two or more bits of agent focus more on message passing and global information on the graph (including nodes and edges). Then the proposed training pipeline is more controllable (through a well-designed reward function) and stable (with supervised learning at the beginning of the training phase to alleviate reward sparsity) than pure reinforcement learning. These differences may be why the accuracy of MCRL is more advantageous in small-scale instances.

We selected the learning curves of MCRL in the first five epochs of the TSP20 training process (the learning curve changes as the time step increases). From Fig. 4b, we can see that learning curves show a significant gradient decrease at the beginning of the first epoch and soon stabilize. At the same time, they become more and more stable in the subsequent epochs and gradually converge to the optimal value. It is known that the training of reinforcement learning is

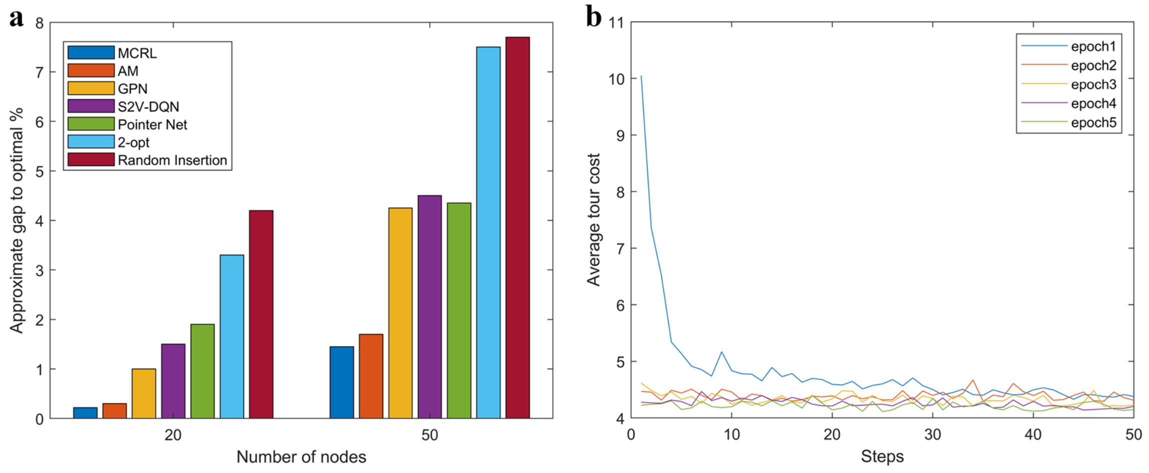


Fig. 4 The left figure **a** shows the approximate gap of different methods relative to the optimal solutions on TSP20/50, where AM [15], GPN [43], S2V-DQN [20], Pointer Network [13] are learning-based methods, and 2-opt and Random Insertion are traditional heuristic algorithms. The right figure **b** shows the learning curves of the first five epochs trained on TSP20

not easy, especially at the beginning of the training phase, which is prone to the reward sparsity problem, and the training of reinforcement learning is not very stable compared to supervised learning.

The results show that applying supervised learning (requiring only a small number of optimal solutions as label data) in the training pipeline for "hot start" is significant. Because the training pipeline gives positive and sufficient feedback to the agent at the beginning, there is no "wandering" and trial and error or the stuck or stagnant phenomenon that often occurs in previous works. Another critical aspect of the stability of the training process is the reward function designed in the Markov decision process for routing optimization. In particular, we elaborated a reward function with global and local horizons combined with the MCTS to control the whole training process jointly and can be effectively integrated with the designed GNN. A critical function of the proposed GNN is to obtain global and local information on routing optimization problems).

To further validate the role of the training pipeline, we removed supervised learning (MCRL without supervised learning as MCRL-RL) and compared the change of the learning curves as the epoch grows on TSP100. Figure 5a shows that MCRL with the training pipeline is more stable and converges faster than MCRL with only reinforcement learning (which is evident at the beginning stage), which proves that the training pipeline can make the training more stable to some extent, making the agent more goal-oriented in finding paths.

The learning curves during training show that the agents in MCRL can effectively interact with the environment and learn the data distribution of the problem instances. Learning-based methods can be tested on new instances as long as they are sufficiently trained, but only if the training and testing sets have similar data distributions, which is the underlying motivation for learning-based methods. So using the trained model for inference and testing on unknown instances is a vital evaluation task. We use the MCRL trained on TSP100 to test on a new TSP100 instance and select the first six epochs to observe how the learning curves change as time increases. Figure 5b shows the inference (test) curves of MCRL. It shows that the learning curves of all six epochs fluctuate and converge in a relatively small range (as the epoch increases). Such volatility is average and reasonable for reinforcement learning. Since the data distribution of the two TSP100 instances is different, the agents need to interact with the new environment interaction to adapt and reason out the correct paths. This demonstrates the efficiency and effectiveness of the proposed training pipeline in testing.

MCTS is one of the core techniques of the AlphaGo family (even many people believe that MCTS is more practical with deep learning than reinforcement learning), and it is also one of the core techniques of MCRL because of its strong ability to traverse and search huge combinatorial spaces. Therefore, a hyperparametric analysis of the MCTS is performed on the TSP50 instance to study the MCTS in MCRL qualitatively and quantitatively. MCTS has two crucial hyperparameters: rollout simulations and search horizons. We first fix the number of rollout simulations of the MCTS to 128 and observe the optimal solutions obtained by MCRL corresponding to different search horizons. Figure 6a shows that MCRL is insensitive to the search horizon of the MCTS, and its performance is better when the search horizon is 6. Then we fix the search range to 6 and observe the variation of the optimal solutions obtained by MCRL when the number of rollout simulations of the MCTS is 64, 128,

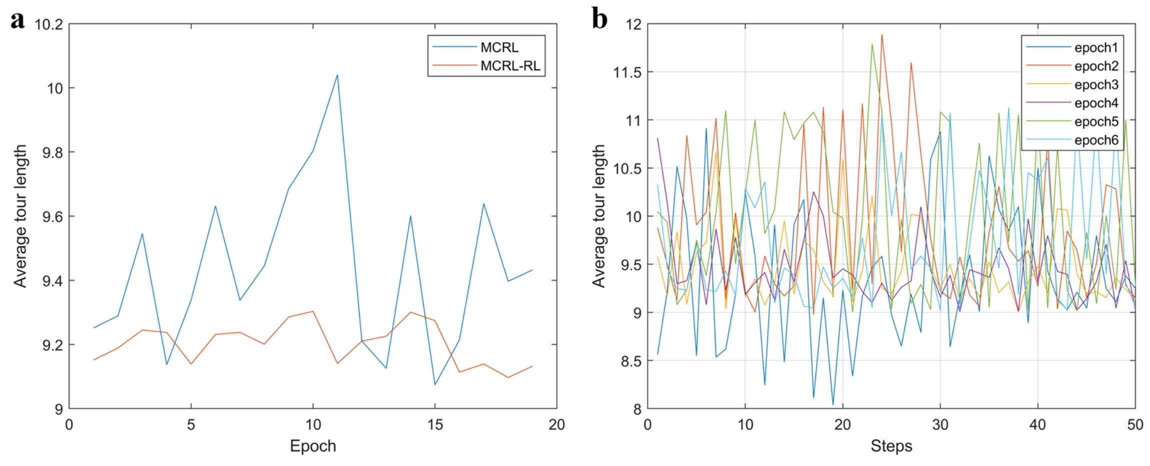


Fig. 5 The left figure **a** shows the average route lengths obtained in the first 20 epochs when MCRL and MCRL without supervised learning (MCRL-RL) are trained on TSP100 (one hour training time). The right figure **b** shows the learning curves of the first six epochs of the trained MCRL (TSP100) tested on the unknown TSP100

and 256, respectively. Figure 6b shows that MCRL's MCTS is sensitive to rollout simulations and performs optimally at many rollout simulations of 128. Adjusting the hyper-parameters has a relatively significant effect on the MCTS, which indicates that the MCTS in MCRL is valuable and practical and plays a relatively significant role in the overall framework.

4.3 Effects on CVRP

As mentioned, MCRL can transform the (basic) VRP into a TSP using a subgraph sampling algorithm with little change in the Markov decision process. The difficulty of VRP is much greater than that of TSP because the former's data distribution is much more complex. On the other hand, previous learning-based approaches purely learn the distribution of VRP instances, which is not sufficient for the complexity of the VRP because we cannot guarantee that the training set is equally distributed with the test set. So it is necessary to pre-process the data using data mining to get a fuller picture of the data distribution (e.g., decomposing a large-scale graph instance into subgraphs of controlled size). It is possible to have multi-agent communication mechanisms (a single agent can also operate independently), so it is necessary to verify the multi-agent role in MCRL. We trained and tested MCRL on CVRP instances with 20, 50, and 100 nodes (with one, two, and five agents, respectively) with different methods (including the solution solver LKH3, the traditional heuristic method RandomCW, PRL (greedy) [21], AM (greedy with sampling) [15], NeuRewriter [25]) (the results of the

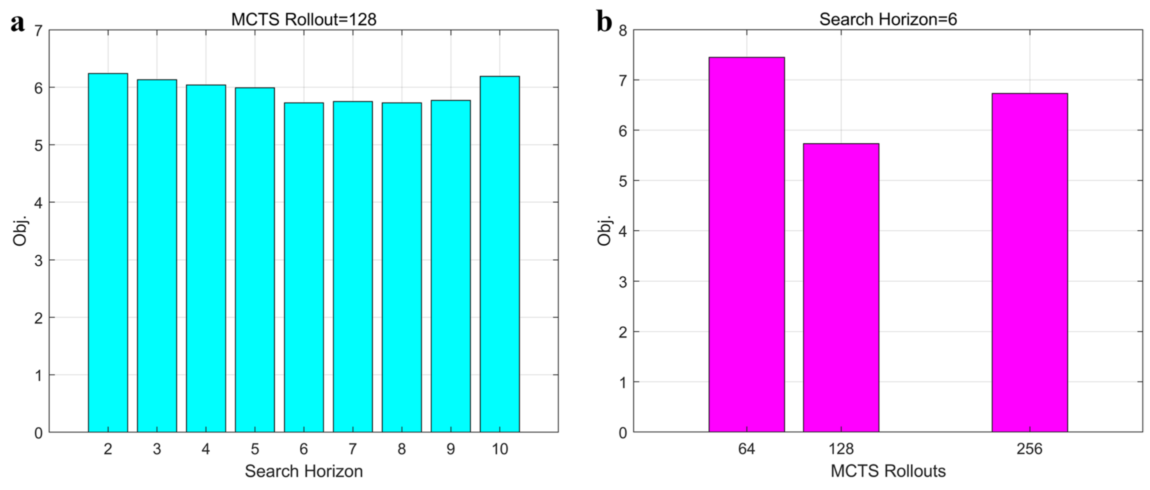


Fig. 6 The left figure **a** shows the MCRL performance changes as the search horizon grows. The right figure **b** shows the MCRL performance changes as the rollout simulations change

comparison methods are taken from previous work [15]). The training times of MCRL on CVRP20, CVRP50, and CVRP100 are 10 min, 30 min, and 1 h, respectively.

Table 1 shows the experimental results of MCRL (with a different number of agents) and other methods. The results show that MCRL with different agents obtains good experimental results regarding solution quality and inference time (MCRL is more advantageous in accuracy than other learning-based methods). The state-of-the-art LKH3 solver has achieved optimal solutions. Still, it is so time-consuming (all in hours) that it can only exist in experiments as a reference baseline or provide some optimal solutions as label data while applying it to real problems (which are more complex and more significant in scale) is still used challenging. The LKH3 solver is based on exact algorithms and improvement heuristics, so this is not fair to methods based on construction heuristics since improvement heuristics often require a given initial feasible solution. NeuRewriter learns the improvement heuristic, which continuously improves the quality of the solution by iteratively improving the initial solution over and over. Although NeuRewriter significantly improves the quality of solutions, testing takes relatively longer because the iterative selection and improvement process consumes more time. The internal operating mechanism of improvement heuristics is to exchange the order of nodes in the solution continuously, that is, to improve the quality of solutions through continuous perturbation and rearrangement. However, this creates a lot of unwanted redundant perturbations.

Instead, the construction heuristics generate solutions from nothing by decoding. Traditional heuristics such as RandomCW are no longer advantageous. After all, they are relatively homogeneous and cannot deal with more complex problems more effectively. But learning construction heuristics shows the flexibility and accuracy that traditional heuristics do not have when faced with complex problems. Moreover, the learning methods based on the construction heuristic require a short testing time because it is easy to test on similar problems as long as the distribution of the problem is fully learned. For example, the state-of-the-art construction heuristic-based learning method AM [107] models problem instances by transformer [17], uses a pointer mechanism to output probability distributions, and uses reinforcement learning with baselines to train decoding policy forming a mainstream paradigm for solving routing problems. The results show that the solutions obtained by AM using the sampling algorithm are of higher quality than the greedy algorithm. Still, it also consumes more time, probably because the sampling algorithm requires more exploration and observation of the data, while the greedy algorithm is more biased in selecting the locally optimal solution.

The results show that the training pipeline with MCTS is more efficient and stable than pure reinforcement learning methods with greedy or sampling than other learning-based construction heuristics. Analyzing the reason, we feel that MCRL's two-stage training pipeline combines the precision of supervised learning with the generalization of reinforcement learning. We only need a small amount of label data to make the supervised training learn a good initial policy, effectively avoiding the problem of sparsity and instability of rewards in the initial stage of reinforcement learning training. The parameters learned from supervised training can also guide subsequent reinforcement learning agents to find and reason more effectively. Besides, MCRL has a reward function that controls the entire framework training, the reward function combined with the A* algorithm fine-tunes the MCTS, which are factors that contribute to the quality of the solution.

The results in Table 1 show that MCRL with one, two, and five agents are competitive with other methods in terms of solution quality and testing time, and the higher the quality of the solutions obtained, the shorter the time required as the number of agents increases. This indicates that the multi-agent and communication mechanism in MCRL is

Table 1 MCRL vs. baselines in CVRP instances

Method	$n=20$			$n=50$			$n=100$		
	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
LKH3	6.14*	0.00%	2 h	10.38*	0.00%	7 h	15.65*	0.00%	13 h
Random CW	6.81	11.64%	-	12.25	18.07%	-	18.96	21.18%	-
PRL (greedy) [21]	6.59	8.03%	-	11.39	9.78%	-	17.23	10.12%	-
AM (greedy) [15]	6.40	4.97%	1 s	10.98	5.86%	3 s	16.80	7.34%	8 s
AM (sampling) [15]	6.25	2.49%	6 m	10.62	2.40%	28 m	16.23	3.72%	2 h
NeuRewriter [25]	6.16	0.48%	22 m	10.51	1.25%	35 m	16.10	2.88%	66 m
MCRL (one agent)	6.14	0.31%	7 m	10.65	2.60%	21 m	16.40	4.79%	55 m
MCRL (two agents)	6.14	0.22%	4 m	10.48	0.96%	16 m	16.03	2.43%	31 m
MCRL (five agents)	**6.14**	**0.15%**	2 m	**10.40**	**0.19%**	9 m	**15.99**	**2.17%**	23 m

"Obj." represents the best solutions obtained by different methods, "Gap" represents the gap between the best solutions obtained and the optimal solutions, and "Time" represents the time required for testing effective for the whole framework, which is equivalent to an ablation study of the multi-agent mechanism and proves that the multi-agent communication mechanism of MCRL is beneficial to enhance the learning when a reasonable number of agents are available. Analyzing the reasons, the multi-agents of MCRL can learn more diverse policies and more effective learning of distributions, and the communication between them can make more local and global information available to the agents. The multi-agent mechanism matches multiple decoders better, and the former can provide the latter with more diverse distributions so that the latter can learn more about decoding policies that improve the solution quality and shortens the testing time. Besides, during the experiments, we also found that the multi-agent can improve the parallel search efficiency (faster training) of the MCTS because the multi-agent can more easily simulate multiple vehicles traversing in the graph and pass messages more efficiently to obtain the global and local views of the graph. This is helpful to improve the accuracy of the solution and reduce the running time, especially for large graph instances.

However, the results show that MCRL is not optimal in test time, but it still exceeds most of the baselines; after all, MCRL contains many complex neural networks that may lose some efficiency while improving the model capability. The best performer in terms of test time is AM (greedy), i.e., AM using the greedy decoding policy [107], while AM (sampling) using sampling decoding is higher than AM (greedy) in terms of solution quality but weaker than the latter in terms of test time, which indicates the possibility of reducing efficiency while improving solution quality.

We further conduct a statistical test using the Friedman and posthoc Nemenyi tests to compare different learning-based algorithms [44, 45]. Table 2 shows the results and average order values of different methods for CVRP of different sizes. Since NeuRewriter requires a given initial solution, its working mechanism is fundamentally different from other methods, so we do not take it as one of the baselines. We expand the CVRP data set to 200 nodes to compare the differences between the approaches entirely.

We first calculate the Friedman detection values by using the following equations:

Table 2 The Friedman and posthoc Nemenyi tests compare different learning-based methods, where "AOV" represents the average order value of different methods on different data sets

DataSet	PRL (greedy)	AM (sampling)	MCRL
CVRP20	6.59	6.25	6.14
CVRP50	11.39	10.62	10.40
CVRP100	17.23	16.23	15.99
CVRP200	19.52	18.69	17.87
AOV	13.68	12.95	12.60

$$\tau_{\chi^2} = \frac{12N}{k(k+1)}\left(\sum\nolimits_{i=1}^{k} r_i^2 - \frac{k(k+1)^2}{4}\right) \tag{38}$$

$$\tau_F = \frac{(N-1)\tau_{\chi^2}}{N(k-1)-\tau_{\chi^2}} \tag{38}$$

where N is the number of data sets, k is the number of algorithms, r_i represents the average order value of the $i-th$ algorithm, r_i follows the normal distribution, and its mean and variance are $(k+1)/2, (k^2-1)/12$, respectively. τ_F follows the F distribution of $k-1$ and $(k-l)(N-1)$ degrees of freedom.

We first calculate τ_F=-3.01 according to Eqs. (38) and (39). τ_F is less than 5.143 when α= 0.05 in Table 3 and 3.463 when α= 0.1 in Table 4. Therefore, we cannot deny the hypothesis that "all algorithms perform equally".

Then, we use the posthoc Nemenyi test to calculate the critical range CD through the following formula:

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{40}$$

It is found in Table 3 that when k = 3, $q_{0.05}$ and $q_{0.10}$ are equal to 2.344 and 2.052, respectively. According to Eq. (40), the critical values CD are calculated to be 1.657 and 1.451, respectively. According to the average order value in Table 2, the gap between PRL and AM, between AM and MCRL, and between PRL and MCRL does not exceed the critical range (Table 5). Therefore, the test results show that the performance of PRL, AM, and MCRL is not significantly different. This phenomenon shows that the performance of MCRL is close to that of the state-of-the-art methods on data sets with similar distributions, which is reasonable, because the learning mechanism of these construction-based heuristic methods is similar, essentially learning the probability distribution of data and then decoding to get the solution sequence. In addition, most of these learning-based methods are carried out through GNN modeling and reinforcement

Table 3 Some Friedman test critical values at α equal 0.05

N	k		
	2	3	4
4	10.128	5.143	3.863
5	7.709	4.459	3.490

Table 4 Some Friedman test critical values at α equal 0.1

N	k		
	2	3	4
4	5.538	3.463	2.813
5	4.545	3.113	2.606

Table 5 Critical values for the two-tailed Nemenyi test

#Classifiers	2	3	4	5	6
$q_{0.05}$	1.960	2.343	2.569	2.728	2.850
$q_{0.10}$	1.645	2.052	2.291	2.459	2.589

learning training. However, MCRL is superior to other methods in accuracy and training stability, proving that the training pipeline, GNN, multi-agent communication mechanism, and other components we designed are superior to previous methods.

5 Conclusion

This paper proposes a novel modeling and training framework for sequential decision problems on graphs. We develop a GNN as an encoder and a multi-agent mechanism into a multi-decoder to enhance the solutions' accuracy and versatility. We design an MDP for routing problems and apply subgraph sampling that converts a (simple) VRP into a TSP to use almost the same MDP. We also design a more feature-rich and comprehensive reward function for reinforcement learning that serves as the core of the training pipeline to integrate the MCTS engine to retrain the supervised policy network. Besides, the reasoned paths can also serve as rules to solve more complex and larger CO problems. We have provided a generic framework where components can be replaced, perhaps with other state-of-the-art models, to improve the framework's overall performance or other types of CO problems, which will be our future work.

Acknowledgements This research was supported by the "Fundamental Research Funds for the Central Universities" under grant 017231161, and the "Xinghai Associate Professor" under grant 02500364.

Data availability The data that support the findings of this study are available upon request. Requests for data access can be directed to Qi Wang following a formal data-sharing agreement.

Declarations

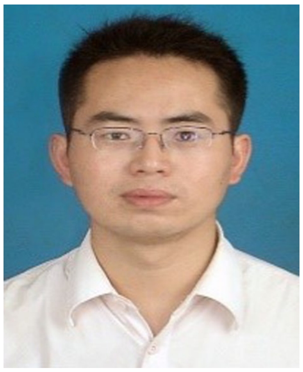
Competing interests The author declared no potential conflicts of interest for this article's research, authorship, and publication.

References

1. Castro-Gutierrez J, Landa-Silva D, Moreno Pérez J (2011) Nature of real-world multi-objective vehicle routing with evolutionary algorithms. Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern, pp 257–264. https://doi.org/10.1109/ICSMC.2011.6083675.
2. Lu H, Zhou R, Fei Z, Guan C (2019) Spatial-domain fitness landscape analysis for combinatorial optimization. Inf Sci (Ny) 472:126–144. https://doi.org/10.1016/j.ins.2018.09.019
3. Niu Y, Shao J, Xiao J, Song W, Cao Z (2022) Multi-objective evolutionary algorithm based on RBF network for solving the stochastic vehicle routing problem. Inf Sci (Ny) 609:387–410. https://doi.org/10.1016/j.ins.2022.07.087
4. Chen C, Demir E, Huang Y (2021) An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots. Eur J Oper Res 294:1164–1180. https://doi.org/10.1016/j.ejor.2021.02.027
5. Windras Mara ST, Norcahyo R, Jodiawan P, Lusiantoro L, Rifai AP (2022) A survey of adaptive large neighborhood search algorithms and applications. Comput Oper Res 146:105903. https://doi.org/10.1016/j.cor.2022.105903
6. Wang Q, Tang C (2021) Deep reinforcement learning for transportation network combinatorial optimization: a survey. Knowl Based Syst 233:107526. https://doi.org/10.1016/j.knosys.2021.107526
7. Ecoffet A, Huizinga J, Lehman J, Stanley KO, Clune J (2021) First return, then explore. Nature 590:580–586. https://doi.org/10.1038/s41586-020-03157-9
8. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, Lillicrap T, Simonyan K, Hassabis D (2018) A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science (80-.) 362:1140–1144. https://doi.org/10.1126/science.aar6404
9. Koster R, Balaguer J, Tacchetti A, Weinstein A, Zhu T, Hauser O, Williams D, Campbell-Gillingham L, Thacker P, Botvinick M, Summerfield C (2022) Human-centred mechanism design with democratic AI. Nat Hum Behav 6:1398–1407. https://doi.org/10.1038/s41562-022-01383-x
10. Fawzi A, Balog M, Huang A, Hubert T, Romera-Paredes B, Barekatain M, Novikov A, Francisco FJ, Schrittwieser J, Swirszcz G, Silver D, Hassabis D, Kohli P (2022) Discovering faster matrix multiplication algorithms with reinforcement learning. Nature 610:47–53. https://doi.org/10.1038/s41586-022-05172-4
11. Wang Q, He Y, Tang C (2022) Mastering construction heuristics with self-play deep reinforcement learning. Neural Comput Appl 35(6):4723–4738. https://doi.org/10.1007/s00521-022-07989-6
12. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, pp 5999–6009
13. Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. In: Advances in neural information processing systems, pp 2692–2700
14. Nazari M, Oroojlooy A, Takáč M, Snyder LV (2018) Reinforcement learning for solving the vehicle routing problem. In: Advances in neural information processing systems, pp 9839–9849
15. Kool W, Van Hoof H, Welling M (2019) Attention, learn to solve routing problems! In: 7th international conference on learning representations, ICLR 2019, pp 1–25
16. Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2021) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32:4–24. https://doi.org/10.1109/TNNLS.2020.2978386
17. Wang Q, Lai KH, Tang C (2023) Solving combinatorial optimization problems over graphs with BERT-based deep reinforcement learning. Inf Sci (Ny) 619:930–946. https://doi.org/10.1016/j.ins.2022.11.073
18. Wang Q, Hao Y, Cao J (2021) Learning to traverse over graphs with a Monte Carlo tree search-based self-play framework. Eng Appl Artif Intell 105:104422. https://doi.org/10.1016/j.engappai.2021.104422
19. Wang Q (2021) VARL: a variational autoencoder-based reinforcement learning framework for vehicle routing problems. Appl Intell 52:8910–8923. https://doi.org/10.1007/s10489-021-02920-3
20. Dai H, Khalil EB, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. In: Advances in neural information processing systems, pp 6349–6359
21. Bello I, Pham H, Le QV, Norouzi M, Bengio S (2019) Neural combinatorial optimization with reinforcement learning. In: 5th

International Conference on Learning Representations, ICLR 2017 - workshop track proceedings, pp 1–15
22. Li Z, Chen Q, Koltun V (2018) Combinatorial optimization with graph convolutional networks and guided tree search. In: Advances in neural information processing systems, pp 539–548
23. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529:484–489. https://doi.org/10.1038/nature16961
24. Xin L, Song W, Cao Z, Zhang J (2021) Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In: 35th AAAI conference on artificial intelligence, AAAI 2021, pp 12042–12049. https://doi.org/10.1609/aaai.v35i13.17430
25. Chen X, Tian Y (2019) Learning to perform local rewriting for combinatorial optimization. In: Advances in neural information processing systems
26. Lu H, Zhang X, Yang S (2018) A learning-based iterative method for solving vehicle routing problems. Iclr 2020. 3, pp 1–13
27. Zheng J, He K, Zhou J, Jin Y, Li CM (2021) Combining reinforcement learning with Lin-Kernighan-Helsgaun algorithm for the traveling salesman problem. In: 35th AAAI conference on artificial intelligence, AAAI 2021, pp 12445–12452. https://doi.org/10.1609/aaai.v35i14.17476
28. Delarue A, Anderson R, Tjandraatmadja C (2020) Reinforcement learning with combinatorial actions: an application to vehicle routing. Adv. Neural Inf. Process. Syst. 2020-Decem
29. Cappart Q, Moisan T, Rousseau LM, Prémont-Schwarz I, Cire AA (2021) Combining reinforcement learning and constraint programming for combinatorial optimization. In: 35th AAAI Conference on Artificial Intelligence, AAAI 2021, pp 3677–3687. https://doi.org/10.1609/aaai.v35i5.16484
30. Zong Z, Wang H, Wang J, Zheng M, Li Y (2022) RBG: hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. Association for Computing Machinery. https://doi.org/10.1145/3534678.3539037
31. Yan D, Weng J, Huang S, Li C, Zhou Y, Su H, Zhu J (2022) Deep reinforcement learning with credit assignment for combinatorial optimization. Pattern Recognit 124:108466. https://doi.org/10.1016/j.patcog.2021.108466
32. Wang Q, Blackley SV, Tang C (2022) Generative adversarial imitation learning to search in branch-and-bound algorithms. In: International conference on database systems for advanced applications. Springer International Publishing, pp 673–680. https://doi.org/10.1007/978-3-031-00126-0_51
33. Hottung A, Kwon Y-D, Tierney K (2022) Efficient active search for combinatorial optimization problems. Iclr 2022, pp 1–10
34. Cai X, Xia C, Zhang Q, Mei Z, Hu H, Wang L, Hu J (2021) The collaborative local search based on dynamic-constrained decomposition with grids for combinatorial multiobjective optimization. IEEE Trans Cybern 51:2639–2650. https://doi.org/10.1109/TCYB.2019.2931434
35. Domínguez-Ríos MÁ, Chicano F, Alba E (2021) Effective anytime algorithm for multiobjective combinatorial optimization problems. Inf Sci (Ny) 565:210–228. https://doi.org/10.1016/j.ins.2021.02.074
36. Yu JJQ, Yu W, Gu J (2019) Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. IEEE Trans Intell Transp Syst 20:3806–3817. https://doi.org/10.1109/TITS.2019.2909109
37. Yin F, Zhao Y (2022) Distributionally robust equilibrious hybrid vehicle routing problem under twofold uncertainty. Inf Sci (Ny) 609:1239–1255. https://doi.org/10.1016/j.ins.2022.07.140
38. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V. (2020) GraphSAINT: Graph Sampling Based Inductive Learning Method. Iclr. 3, pp 415–422
39. Menda K, Chen YC, Grana J, Bono JW, Tracey BD, Kochenderfer MJ, Wolpert D (2019) Deep reinforcement learning for event-driven multi-agent decision processes. IEEE Trans Intell Transp Syst 20:1259–1268. https://doi.org/10.1109/TITS.2018.2848264
40. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, Choi DH, Powell R, Ewalds T, Georgiev P, Oh J, Horgan D, Kroiss M, Danihelka I, Huang A, Sifre L, Cai T, Agapiou JP, Jaderberg M, Vezhnevets AS, Leblond R, Pohlen T, Dalibard V, Budden D, Sulsky Y, Molloy J, Paine TL, Gulcehre C, Wang Z, Pfaff T, Wu Y, Ring R, Yogatama D, Wünsch D, McKinney K, Smith O, Schaul T, Lillicrap T, Kavukcuoglu K, Hassabis D, Apps C, Silver D (2019) Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575:350–354. https://doi.org/10.1038/s41586-019-1724-z
41. Liu A, Chen J, Yu M, Zhai Y, Zhou X, Liu J (2020) Watch the unobserved: a simple approach to parallelizing Monte Carlo Tree Search. Iclr, pp 1–21
42. Wang J, Wu N, Zhao WX, Peng F, Lin X (2019) Empowering A* search algorithms with neural networks for personalized route recommendation. Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min, pp 539–547. https://doi.org/10.1145/3292500.3330824
43. Xin L, Song W, Cao Z, Zhang J (2021) Step-wise deep learning models for solving routing problems. IEEE Trans Ind Inform 17:4861–4871. https://doi.org/10.1109/TII.2020.3031409
44. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
45. Wang H, Yu D, Li Y, Li Z, Wang G (2018) Multi-label online streaming feature selection based on spectral granulation and mutual information. Springer International Publishing.https://doi.org/10.1007/978-3-319-99368-3_17

Qi Wang received his B.S., M.Eng., and Ph.D. degrees in software engineering from Jilin University (Changchun city), Central South University (Changsha city), and Fudan University (Shanghai) in 2012, 2016, and 2022, respectively, in China. Currently, he holds the position of lecturer at the Information Science and Technology College at Dalian Maritime University. His research interests encompass combinatorial optimization, deep learning, and reinforcement learning.



Yongsheng Hao received his MS Degree of Engineering from Qingdao University in 2008. Now, he is a senior engineer of Network Center, Nanjing University of Information Science & Technology. His current research interests include cloud computing, distributed and parallel computing, mobile computing, Grid computing, web Service, particle swarm optimization algorithm, and genetic algorithm. He has published more than 50 papers in international conferences and journals.