Neural Computing and Applications (2023) 35:4723–4738
https://doi.org/10.1007/s00521-022-07989-6

ORIGINAL ARTICLE

Mastering construction heuristics with self-play deep reinforcement learning

Qi Wang[1] · Yuqing He[2] · Chunlei Tang[3]

Received: 8 December 2021 / Accepted: 20 October 2022 / Published online: 29 October 2022

Abstract
Learning heuristics without expert experience to construct solutions automatically has always been a critical challenge of combinatorial optimization. It is also the pursuit of artificial intelligence to construct an agent with the planning ability to solve multiple problems simultaneously. Nonetheless, most current learning-based methods for combinatorial optimization still rely on artificially designed heuristics. In real-world problems, the environment's dynamics are often unknown and complex, making it challenging to generalize and implement current methods. Inspired by AlphaGo Zero, we propose a novel self-play reinforcement learning algorithm (CH-Zero) based on the Monte Carlo tree search (MCTS) for routing optimization problems in this paper. Like AlphaGo Zero, CH-Zero does not require expert experience but some necessary rules. However, unlike other self-play algorithms based on MCTS, we have designed offline training and online reasoning. Specifically, we apply self-play reinforcement learning without MCTS to train offline policy and value networks. Then, we apply the learned heuristics and neural network combined with an MCTS to make inferences on unknown instances. Since we did not incorporate MCTS during training, this is equivalent to training a lightweight self-playing framework whose learning efficiency is much higher than the existing self-play-based methods for combinatorial optimization. We can employ the learned heuristics to guide MCTS to improve policies and take better actions at runtime.

Keywords Combinatorial optimization · Deep learning · Reinforcement learning · Monte Carlo tree search

1 Introduction

Optimization problems determine the optimal configuration or "value" among several options [1, 2]. They generally fall into two categories: configurations with continuous or discrete variables. A continuous optimization problem is, for instance, finding solutions to a convex programming problem, whereas a discrete optimization problem is finding the shortest route among all paths in a graph. The boundary between the two might be challenging to establish at times. Since its solution rests in a limited vertices collection of the convex polytope, the linear programming job in continuous space may be treated as a discrete combinatorial problem, as proved by Dantzig's method [3].

Routing problems, such as the traveling salesman problem (TSP) [4, 5] and vehicle routing problem (VRP) [6, 7] (Fig. 1), are a type of combinatorial optimization (CO) problem with a wide range of real-world applications. Routing problems are NP-hard in general due to their combinatorial character. The intriguing thing about these issues is that they appear to be straightforward and beneficial, yet finding a universal and effective solution may be difficult [8]. Exact methods, such as branch-and-bound algorithms, provide a solid theoretical assurance of optimality, but the (worst-case) computing complexity is exponential. On the other hand, approximate algorithms guided by heuristics may yield near-optimal solutions with polynomial processing complexity and are thus often used, particularly for large-scale issues. On the other hand, traditional approaches depend too heavily on artificially

✉ Qi Wang
17110240039@fudan.edu.cn

[1] School of Computer Science, Fudan University, Shanghai, China

[2] Institute for Data Industry, Fudan University, Shanghai, China

[3] Brigham and Women's Hospital, Harvard Medical School, Boston, USA

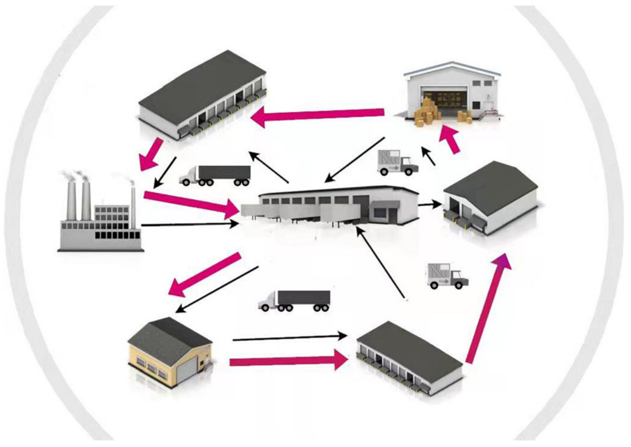

Fig. 1 An example diagram of VRP and TSP. The purple arrow line indicates the routing of a complete TSP, and the black arrow line indicates the routing of a VRP. Different groups of customer nodes represent different data distributions, and the goal of learning-based approaches is to learn these potential distributions

developed heuristics. More significantly, they are all directed at specific situations, and if the problem statement changes slightly, we have to redesign the corresponding algorithm.

As one of the most contemporary artificial intelligence techniques, machine learning [9] (e.g., deep learning [10] and reinforcement learning [11]) learns heuristics from data samples. Most machine learning algorithms use an encoder–decoder structure to learn construction heuristics by continuously adding nodes to an empty or partially solved problem until complete [12]. At each construction stage, the encoder encodes node information into feature embeddings, and the decoder predicts the odds of picking each valid node. Different approaches (such as sampling or beam searching) are used to produce a set of solutions from the training building strategy to choose the best one. Because games are abstractions of many real-world situations, they are frequently used as AI benchmarks. In perfect-information games, significant progress has been made. AlphaGo series (including deep learning, reinforcement learning, and Monte Carlo tree search [13]) [14–18], for example, have achieved state-of-the-art performance in the Go game. In imperfect information games, agents compete or collaborate with others in a partially visible world, have been the focus of recent studies. The game may be viewed as a combinatorial optimization issue; thus, it is interesting investigating whether incorporating techniques from the cutting-edge AlphaGo series into classic CO might also provide positive results.

One of the significant difficulties of artificial intelligence is repurposing and reusing previously learned skills to learn more sophisticated skills or solve new issues. As Bengio et al. [7] noted, we anticipate that the information gained in one environment may be transferred to other contexts with different but perhaps related distributions and attain optimal generalization when the training and test data originate from the same distribution. The majority of today's learning-based approaches are simply learning data distributions. The trained model may quickly make inferences when the test and training sets have the same distribution. Nonetheless, they rely too heavily on data distribution and need the use of heuristics that have been intentionally constructed. Although self-play approaches based on AlphaGo Zero generate data labels to enable self-training, they frequently rely too much on processing resources (GPU, TPU, etc.) and training time. This is because the reinforcement learning method, when paired with tree search, necessitates continual repetition and recursion, resulting in inefficient training.

This paper proposes a new self-play reinforcement learning algorithm for learning constructive heuristics. Unlike all previous methods based on AlphaGo Zero, we did not incorporate Monte Carlo tree search (MCTS) [13] in self-play learning but applied MCTS on unseen data to assist inference. To mitigate the performance degradation when learning solutions, we design a scheme to bridge the gap between online search and offline learning. Our approach can solve many offline and online problems with unknown priors. This approach benefits from the training strategy's generalization ability, real-time performance, and robustness obtained from online searches. Specifically, we employ a graph neural network to construct a parameter-sharing deep neural network, including policy and value networks. We employ a pointer recurrent neural network to store the state and act as a central axis to link policy and value networks. We apply the DQN algorithm [19] to update the Q-value to make the policy network self-learn the heuristic. Then we only utilize the MCTS as a heuristic search algorithm for the decoding reasoning stage. In summary, the contributions of this paper are as follows:

- We applied the GNN to design a novel policy and value network architecture for parameter sharing.
- We designed a pointer recurrent neural network based on LSTM to memorize the state, omit pre-training, and solve the partially observable Markov decision process problem.
- We designed a novel self-play learning method for offline training and online reasoning. We did not add MCTS in training but utilized it as a heuristic search and combined it with the learned heuristics to reason during the test, which can effectively improve training efficiency and enhance generalization.

2 Related work

Applying machine learning (ML) to combinatorial optimization (CO) problems has been a rapid development direction in recent years [20]. Although CO problems are usually difficult to solve on a large scale, ML algorithms can approximate these problems [21]. Compared with traditional methods, learning-based methods have many advantages, such as better scalability, generalization ability, and versatility. There are three dominant approaches for solving the CO problem [22], a traditional algorithm, which is non-ML, an end-to-end application of ML techniques, and some work that combines both of these approaches in an enhanced manner. Combining heuristic search algorithms and ML can benefit each other by using search to accelerate learning or learn better models for use in search. We will introduce traditional algorithms, learning-based methods, and AlphaGo Zero-based methods to make this paper self-consistent.

2.1 Traditional algorithms

2.1.1 Exact algorithm

It searches the entire solution space and can guarantee to find the optimal solution. If the search space is too large, we generally diminish the search space by pruning and other methods [23]. A common approach is to model the problem as integer programming (IP) or mixed-integer programming (MIP) [23], including branch-and-bound, branch-and-cut, and constraint programming. Branch and bound [24] essentially constructs the search space in the form of a tree and continuously selects and splits nodes by combining pruning to find the optimal solution efficiently [25]. The industry has produced solution solvers such as Concorde especially optimized for TSP based on this. An accurate algorithm needs to find the optimal solution, but it is not suitable for large-scale problems due to the nature of the NP-hard.

2.1.2 Approximation algorithm

It is not looking for the optimal solution but the sub-optimal solution or near-optimal solution, but its advantage is that there will be a sure optimality guarantee [26]. Specifically, it can ensure that the solution given in the worst case is not inferior to a certain multiple of the optimal solution, called the approximation ratio. Although there is no polynomial complexity algorithm for the optimal solution, many polynomial complexity algorithms exist for the near-optimal solution. Approximation algorithms include sequential, greedy, dynamic, randomized, and primary-dual methods [27, 28].

2.1.3 Heuristic algorithm

The heuristic algorithm is essential to find an approximate solution. Compared with the approximation algorithm, it has no theoretical guarantee of optimality, but generally speaking, it can find a better solution than the approximation algorithm. For example, a heuristic algorithm constructs a solution based on the heuristic related to the problem, such as the nearest neighbor and minimum spanning tree. One is local search, which modifies the intermediate solution according to the heuristic and iteratively searches the solution space [9]. Other methods adopt meta-heuristic search frameworks [29], such as genetic algorithm, simulated annealing, and tabu search. The state-of-the-art heuristic algorithm for TSP is Lin–Kernighan–Helsgaun (LKH) [30, 31], and the scale of TSP it can solve can reach tens of thousands. Nevertheless, the time consumed by the heuristic algorithm for large-scale problem instances is enormous.

2.2 Learning-based methods

We generally divide the learning-based methods into attention-based and graph neural network (GNN)-based methods, although some overlap and fuse.

2.2.1 Attention-based methods

Inspired by the seq2seq model [32] in natural language processing (NLP), the pointer network [33] is proposed to solve the CO problem in a targeted way. It combines seq2seq and attention to cope with the output sequence's length depending on the input sequence's length in CO problems. The probability of each node in the input sequence of the pointer network will be obtained by outputting the predicted attention.

Bello et al. [34] are the first to clarify that reinforcement learning is more suitable for CO problems than supervised learning. It is expensive or impossible to obtain lots of label data and apply the actor-critic algorithm [35] to train the pointer network to solve the TSP problem.

Based on [33, 34], Nazari et al. proposed an end-to-end RL method to solve VRP, simplifying the pointer network and effectively handling static and dynamic elements. It extends the pointer network to VRP and TSP and exceeds traditional heuristics and OR-tools [36].

Kool et al. built a framework based on the transformer utilizing only the attention mechanism [37] to solve the combinational optimization problem [38]. Instead of using the sequence as input, they utilize graphs as input, eliminating the dependence on the order of the nodes in the input [39]. No matter how we arrange the nodes, the output will not vary if the given graph does not alter, advantageous over the sequential approach.

2.2.2 GNN-based methods

As a generalization of deep learning in graph data, a graph neural network (GNN) [40, 41] has been widely utilized in CO in recent years due to its strong ability to represent the intrinsic structure of graphs and its ability to aggregate information of nodes and edges.

Dai et al. proposed finding evaluation functions with graph embedding [42]. They first applied Structure2Vec to embed the graph into the low-dimensional space and then employed the Q-learning algorithm and the greedy heuristic searches to add nodes to solve the problem [43].

Li et al. applied supervised learning training GCN [44] to guide the parallel tree search process, quickly generating many candidate solutions and selecting one after subsequent optimization [45]. Mittal et al. designed a two-stage training method based on [43, 45], in which supervised learning followed by RL and greedy probability distribution was applied to solve CO problems on large-scale graphs [46]. The methods for CO, which are designed into an encoder–decoder paradigm based on the GNN framework, also include [47–50].

Chen et al. proposed a neural writer that learns a policy to select heuristic algorithms and rewrites local components of the current solution to improve it until it converges [8] iteratively. Lu et al. proposed “L2I,” which proceeds from a random initial solution and learns to iteratively improve the solution with an improved operator selected by the controller based on RL [51].

2.3 AlphaGo Zero-based methods

AlphaGo Zero [17] can already achieve the ultimate goal of board games. Given only the game rules, the agent starts from the initial random state and surpasses previous human experts and AlphaGo [14] through continuous self-play. And AlphaGo Zero has been applied to other chess games to acquire the same excellent effect. The AlphaGo Zero algorithm consists of reinforcement learning, deep learning, and Monte Carlo tree search. The core idea of AlphaGo Zero is policy iterative reinforcement learning based on neural networks, that is, finally learning a policy network. Its input is a particular chessboard position, and its output is the probability distribution of the moves that can be placed in this position.

2.3.1 CO versus GO

(1) CO also has a vast solution space like Go. For example, the solution set of TSP with n cities can reach an order of magnitude of $O((n-1)!)$ [8]. The number of states and the average number of moves in the Markov decision process (MDP) of the medium-scale graph coloring problem exceed chess and Go [52]. We can also create an appropriate mathematical model for CO problems and adopt the MCTS to diminish the solution space. (2) Like winning and losing rules in Go, the CO problem also has a clear objective function and constraint conditions (rules) to evaluate current policy and learn from reward signals. (3) The heuristic function based on expert experience or the model trained with small-label data may be locally optimal in the convergence domain. Hence, we can achieve self-optimization by generating data and high-quality sample labels by self-play like AlphaGo Zero (or AlphaZero [53]).

Some works have been applying AlphaGo Zero’s core ideas to combinatorial optimization. Laterre et al. [54] combined learnable policies and value functions with MCTS and self-play. To solve the 2D and 3D boxing problems in single-player games, they built the best solution by adding a ranking reward mechanism, which reshaped the rewards based on the relative performance of recent games. It aims to provide a natural course for a single agent, similar to a natural opponent in a two-person game. Huang et al. [52] utilized the policy and value network architecture to solve the graph coloring problem of large graphs. They used data mining techniques to decrease the MCTS space to a controllable range to learn graphs with millions of vertices. Abe et al. [55] proposed using the graph neural network to solve the state’s variable size representation in the search tree and modify the AlphaGo Zero algorithm by normalizing the value function.

3 Methodology

3.1 Problem setup and preliminaries

The learning-based method predicts the distribution of unknown data by learning the given data distribution. Compared with traditional algorithms, it has more substantial generalization and can usually achieve one method for multiple problems. This article focuses on applying deep reinforcement learning to solve the path optimization problem on the graph, namely the classic vehicle routing problem (VRP) and the traveling salesman problem (TSP). TSP is a simple case of VRP. (We can think of TSP as a VRP that only returns to the starting node after a vehicle traverses all customer nodes once.)

We can describe VRP (Fig. 1) in detail as follows: Given a warehouse, there are k vehicles with a load capacity of q_k to deliver goods to n customer nodes. The demand for the ith customer point is d_i. The vehicles transport the goods to various customer points and then return to the warehouse. We aim to arrange vehicle routes rationally to meet customer needs and make the shortest tour. Based on the above constraints, we establish the following mathematical model with the shortest driving path as the objective function:

$$\min D = \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{s}^{k} c_{ij}x_{ijs} \tag{1}$$

$$\sum_{i=0}^{n} x_{ijs} = y_{js}, \quad j = 1,2,\ldots,n; \quad \forall s \tag{2}$$

$$\sum_{j=0}^{n} x_{ijs} = y_{is}, \quad i = 1,2,\ldots,n; \quad \forall s \tag{3}$$

$$\sum_{i=0}^{n} d_i y_{is} \le q_k, \quad \forall s \tag{4}$$

$$\sum_{s=1}^{k} y_{is} = \begin{cases} 1, & i = 1,2,\ldots n \\ k, & i = 0 \end{cases} \tag{5}$$

here Eq. (1) is the objective function, c_{ij} represents the transportation cost from customer point i to customer point j. $x_{ijs} \in \{0,1\}$ is the decision variable that represents whether the vehicle s (s is the vehicle number and $s \in \{1,2,\ldots,k\}$) driving from node i to node j. $y_{is} \in \{0,1\}$ is also a decision variable, which indicates the route choice of a single customer to the next customer. Equations (2) and (3) indicate that each customer node must be visited (in) once and must be left (out) once, respectively. Equation (4) indicates that the carrying capacity of all vehicles can cover the total demand of all customer nodes. Equation (5) indicates that each vehicle departs from node 0, and the remaining nodes can only be passed by one vehicle.

Since we need to face the graph data that the deep reinforcement learning algorithm can process, we formally define the path optimization problem and the corresponding Markov decision process. We define $G(V,E)$ as an undirected and unlabeled graph, where V is a set of nodes and E is a set of edges. We define the path optimization problem as a three-tuple $(I,\overline{S},f)$, where I represents the set of problem instances, $\overline{S}$ represents the feasible solution set of the problem instances, and f represents the objective function that maps the solution in $\overline{S}$ to real values. We adopt the Markov decision process to model the sequence solution process to employ reinforcement learning. As the time step t increases, a partial solution continues to expand and finally forms a complete feasible solution, specifically:

3.1.1 States

The state $s_t \in \overline{S}$ at time step t is an ordered sequence of traversal nodes on the graph G, and we apply graph neural network (GNN) [56] to encode each state as a vector in an l-dimensional space. When we traverse all the nodes on the graph, we reach the end state.

3.1.2 Actions

An action a is a node on the graph G that we have not traversed yet.

3.1.3 Transition

Transition is deterministic in the path optimization problem. It means adding the selected node $v_i \in V$ in $\widehat{s}$ to s, where s and $\widehat{s}$ indicate the sequence of traversed nodes and the sequence of nodes that have not been traversed. We define the transition probability as $p(\widehat{s}|s_t,a)$.

3.1.4 Rewards

As long as the feasible solution we obtain expands, we will reward the agent. We employ the following equation to determine the specific reward value:

$$r_{t+1} = r(s_t,a_t) = f(s_{t+1}) - f(s_t) \tag{6}$$

where f represents the objective function mapping the partial solution s_t to real values, i.e., the more partial solutions obtained, the more rewards the agent gets. Suppose the reward is defined only as 0 or 1. In that case, it equals the resulting solution value at the end of the episode, and zero anywhere else would make the learning process more challenging for its sparsity. VRP and TSP are both minimization problems; therefore, each time t, there is a cost, or a negative reward, which is defined as the difference in the objective value for the partial solution obtained by extending the present partial solution by taking a_t. This is just the distance traveled between steps t and $t+1$ in path optimization problems. Our reward function has better generalization and can be applied to many combinatorial optimization problems, as long as its solution set can be added like a greedy algorithm.

3.1.5 Policy

We aim to find a policy $\pi(a|s)$ to learn the distribution of actions, that is, to learn a heuristic policy to guide the agent to take actions (select nodes). It leads to the distribution on the trajectory $\rho = (s_t,a_t,r_{t+1})_{t=0,\ldots,T-1}$ (T is the maximum number of time steps):

$$p(\rho) = p(s_0)\prod_{t=0}^{T-1}\pi(a_t|s_t)p(s_{t+1}|s_t, a_t) \tag{7}$$

For each s that appears in the trajectory, we calculate and update $Q(s,a)$ and average all samples related to s, a. Our ultimate goal is to find an optimal heuristic policy $\pi^*(a|s)$ through Q-learning [57], the model can make inferences over unseen data.

$$\pi^*(a|s) = \mathrm{argmax}_a Q(s,a) \tag{8}$$

where $Q(s,a)$ is the Q-function.

3.2 Neural network architecture

Our network architecture consists of five modules (as illustrated in Fig. 2): (policy) actor-network encoder, (value) critic-network encoder, LSTM interpreter, policy network decoder, and value network decoder. Since most of the above components are universal, our network architecture is suitable for multitask learning.

The most mainstream deep learning method for the graph data structure is the graph neural network (GNN) [56]. We can use it to model graph instances and obtain graph structure information (including global and local information) through aggregation and message passing. Also, the agent can learn the probability distribution of feasible solutions to avoid blindly finding paths without considering the order of nodes. We apply GN [40], a variant of message passing GNN, as the encoder of the policy network to extract features from the graph. We represent each node in the graph with a feature vector and merge its neighbor nodes' information according to its topological structure. We employ the edge information to update the node information to generate a new node vector representation. Besides, we also utilize global weights as extra features, specifically:

$$e_i^{(t+1)} = \varphi_e\left(e_i^{(t)}, \rho^{v\to e}(V), w\right), \quad \forall e_i \in E \tag{9}$$

$$v_i^{(t+1)} = \varphi_v\left(v_i^{(t)}, \rho^{(e,v)\to v}(V,E), w\right), \quad \forall v_i \in V, \tag{10}$$

$$w^{(t+1)} = \varphi_w\left(w^{(t)}, \rho^{(e,v)\to w}(V,E)\right) \tag{11}$$

here $\rho^{v\to e}$, $\rho^{(e,v)\to v}$, $\rho^{(e,v)\to w}$ are aggregation functions of the information collected from the graph, φ_e, φ_v, φ_w are neural networks that update the feature vectors of nodes and edges, respectively, and w is the global feature vector.

In the natural environment, the complete state of the system is rarely provided to the agent or even determined. The partially observable Markov decision process (POMDP) [58] better captures the dynamics of many real-world environments by explicitly acknowledging that the feeling received by the agent is only a partial glimpse of the state of the underlying system.

AlphaGo [14] applies a training strategy of supervised learning followed by reinforcement learning to make the model hot-start, but this still relies on expert experience and label data. Some previous work based on reinforcement learning, such as GCOMB [46], applies supervised reinforcement learning to pre-train the model like AlphaGo, reducing the generalization and may overfit the given label data. Bello et al. [34] designed a two-stage reinforcement learning strategy, using reinforcement learning to fine-tune the model trained with reinforcement learning. Multi-stage training is likely to cause cascading errors, while end-to-end training has no such problem.

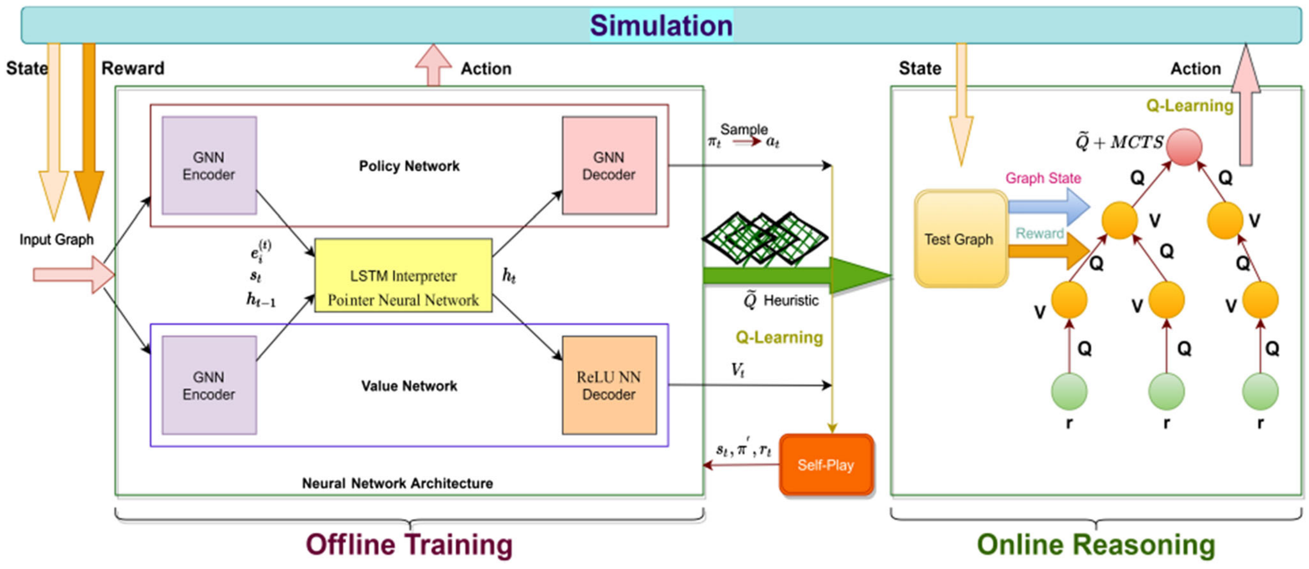

Fig. 2 The complete process of our method, in which the left is offline training and the right is online inference, and the states and actions are transferred through MCTS simulation to update each other

Thus, we have designed a new memory-based end-to-end neural network architecture, which applies LSTM in the training phase to omit pre-training by remembering historical trajectory sequences.

Parameter sharing is crucial for policy and value networks because it can promote the sampling efficiency of reinforcement learning. We employ LSTM as the core axis to combine the policy and value networks to form a linkage mechanism. As long as the parameters of one network are updated, the other network will be updated accordingly. In this way, we can only learn the parameters of one network to train the entire network architecture.

To better share the parameters of the policy network and the value network, we apply the same GNN as the policy network encoder and model the encoder of the value network. We apply the GNN of the policy network encoder to encode the node features into the state s_t and then encode the state into the latent memory vector h_t. We propose a new pointer memory cell to develop a series of p-dimensional latent memory states. Then the policy network converts the output of the LSTM into a probability vector π on the action space, and the policy network utilizes the output to estimate the value function V. For simplicity of description, we omit their shared parameters θ, as follows:

$$s_t = \phi_{enc}(e_i^{(t)}) \tag{12}$$

$$h_t = \phi_{LSTM}(s_t, h_{t-1}) \tag{13}$$

$$P(s_t, a_t) = \pi_t = \phi_{actor}(h_t) \tag{14}$$

$$Q(s_t, a_t) = V_t = \phi_{value}(h_t) \tag{15}$$

The ϕ_{enc}, ϕ_{LSTM} in Eq. (12), (13) is our pointer recurrent neural network, which we employ LSTM cells to describe, specifically:

$$F_t^{(i)} = sigmoid(W_f \cdot \left[h_{t-1}, e_i^{(t)}\right] + b_f) \tag{16a}$$

$$I_t = sigmoid(W_i \cdot \left[h_{t-1}, e_i^{(t)}\right] + b_i \tag{16b}$$

$$\widetilde{C}_i = \tanh(W_C \cdot \left[h_{t-1}, e_i^{(t)}\right] + b_C) \tag{16c}$$

$$C_i^{enc} = F_t * C_{i-1}^{enc} + I_t * \widetilde{C}_i \tag{16d}$$

$$O_t = \sigma(W_o \cdot \left[h_{t-1}, e_i^{(t)}\right] + b_o) \tag{16e}$$

$$h_t = o_i * \tanh(C_i^{enc}) \tag{16f}$$

where $*$ is element multiplication, W_f, W_i, W_C, W_o are model weight parameters, and b_f, b_i, b_C, b_o are model bias parameters. We feed the node state vector obtained by GNN to the pointer recurrent neural network, and the GNN and pointer recurrent neural network are also updated and iterated simultaneously.

The decoder of the policy network utilizes the same graph neural network as the encoder. Our value network includes three modules: a GNN network, a pointer recurrent neural network shared with the policy network, and a two-layer ReLU neural network decoder. The value network has the same encoder structure as the policy network and encodes the input graph into potential memory states and hidden states. After the pointer neural network, we decode the obtained hidden state into a self-play baseline prediction (scalar v) using two fully connected neural networks with d and 1 unit, respectively.

3.3 Offline training

Our technique consists of two phases: First, we train the agent offline in a non-policy way in a simulated environment using a graph neural network to learn a state representation. Second, we employ the offline learned policy as a Monte Carlo tree search (MCTS) heuristic during testing and inference. This method allows us to train a suitable offline policy while ensuring a robust online tree search solution.

Since it is challenging to obtain label data, we also adopt a method similar to the data generator in AlphaGo Zero [17] to generate data for reinforcement learning. We do not need expert experience, but we need to set some rules for generating data. The available context of the policy network in each state is the node's position in the routing problem instance, the distance between the nodes, the sequence of nodes in the partial route, and the set of remaining nodes. We can utilize the above to construct a particular graph to feed the policy network. Due to the property of GNN, which exploits edge information to update node information to generate a new node representation, we construct edges that cannot be utilized in the final solution. We define a feature vector on each node, which contains the node coordinates, an indicator of whether it is in the partial tour, and the distance between nodes and their neighbors as the weight of edges.

Unlike AlphaGo Zero or methods for routing problems based on AlphaGo Zero, we do not incorporate Monte Carlo tree search (MCTS) when applying reinforcement learning to train the network, because the use of reinforcement learning with MCTS to train neural networks often consumes many computing resources. We designed a lightweight AlphaGo Zero-like framework to develop offline training and online inference mode. Specifically, we employ reinforcement learning without MCTS to train the network offline and then the trained neural network to make online inferences on the test data set.

We design a training pipeline to train the parameter θ from beginning to end, and we further define $f_\theta(s_t) = (P(s_t, a_t), V(s_t))$ according to AlphaGo Zero, where $P(s_t, a_t)$ is the action probability, and $V(s_t)$ is the evaluation score of the model predicted by the network f with parameter θ for the given data set D, and state s_t. The neural network predicts the probability of taking one action, which addresses a given task on a data set. The network input is a self-playing training instance (s_t, π', r_t), where r_t is the evaluation of the pipeline at the end of the traversal. The parameter θ of network f is adjusted by the stochastic gradient descent (SGD) of loss function L, which calculates the sum of the mean squared error and the cross-entropy loss:

$$L(\theta) = (r - v)^2 - \pi'^T logP + c\|\theta\|^2 \tag{17}$$

The L2 regularization parameter c is utilized to prevent overfitting. The network output is the probability of action $P(s_t, a_t)$ and the evaluation $V(s_t)$ of pipeline performance.

AlphaGo Zero [17] runs multiple MCTS simulations to generate a set of trajectories with more positive rewards, which can be understood as being generated by a promoted policy π', while learning these trajectories can, in turn, promote the policy. But we adopt an offline training method without MCTS, and these trajectories are discrete rather than continuous. Hence, we build the offline part of the method based on the off-policy DQN algorithm [57, 59] instead of REINFORCE [35] to update the Q-network, in which we learn a neural approximation $\widetilde{Q}(s, a; \theta) \approx Q(s, a)$ parameterized by θ:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s, a)] \tag{18}$$

Since the Q-network (value Network) and the policy network share the same set of parameters, the policy network will be promoted simultaneously as soon as the Q-network (value network) parameters are updated. The offline learning algorithm is demonstrated in Algorithm 1.

3.4 Online reasoning

In the testing or running phase, we apply an online Monte Carlo tree search (MCTS) to optimize the decision-making in the real-time observed state. We apply offline trained networks and $\widetilde{Q}$ value, combined with MCTS to evaluate all appropriate actions in the current state and output the optimal action in this state within the maximum time step (Fig. 3).

Our algorithm runs multiple simulations in the MCTS [13] through neural network predictions $(P(s_t, a_t), V(s_t))$ using them to search for better evaluation. The search results promote the predicted results given by the network by using update rules to improve network policy:

$$U(s_t, a_t) = Q(s_t, a_t) + cP(s_t, a_t)\frac{\sqrt{N(s_t)}}{1 + N(s_t, a_t)} \tag{19}$$

where $Q(s_t, a_t)$ is the expected reward of an action a_t starting from state s_t, $P(s_t, a_t)$ is the neural network's estimation of the probability of an action a_t starting from state s_t, $N(s_t, a_t)$ is the number of an action a_t starting from state s_t, $N(s_t)$ is the number of times to access the state s_t, and c is the constant to determine the amount of exploration. At each step of the simulation, we need to find the action a_t and state s_t that maximize $U(s_t, a_t)$, and add the new state to the tree if it does not exist in the neural network estimation $(P(s_t, a_t), V(s_t))$; otherwise, the search is called recursively.

We developed a reverse link algorithm that uses MCTS to leverage Markov decision process (MDP) transfer functions. That is, in each MCTS simulation, the trajectory is expanded by selecting actions based on variations of the PUCT algorithm [17] from the root state s_0:

Algorithm 1 Self-Play Training Algorithm

```
Require: Generated graph $G$, hyper-parameters $M$, $N$ relayed to fitted Q-learning,
Maximum episodes $MaxEpisode$, sample size $T$
1.  Initialize experience replay memory $M$ to capacity $N$
2.  for $episode = 0,1,\ldots, MaxEpisode$ do
3.      Draw graph $G$ from distribution $D$
4.      for $t = 1, \ldots, T$ do
5.          $\pi^*(a|s) = argmax_a Q(s, a)$ // Choose the best action from the policy
6.          Select a node to add to the partial solution set: add $v_i$ in $\hat{s}$ to $s$
7.      end for
8.          Add tuple $(s_t, a_t, r_t)$ to $M$, $t = 1, 2, \ldots, T$
9.          Sample random batch from $B$ form $M$
10.         $L(\theta) = (r - v)^2 - \pi^T logP + c\|\theta\|^2$ // Equation (17)
11.         Update $\theta$ by Adam over the equation (17) for $B$
12. end for
13. return $\theta$
```

$$a_t = \mathrm{argmax}_a \left\{ \beta \cdot \frac{\pi_\theta(a|s_t)^\mu \sqrt{\sum_{a'} N(s_t, a')}}{1 + N(s_t, a)} + \frac{W(s_t, a)}{N(s_t, a)} \right\} \quad (20)$$

where β and μ are constants that resemble γ.

When training with AlphaGo Zero [17], the data generator repeatedly generates graph instances as self-play records based on the current best model of MCTS, which is referred to (s_t, π_θ, r_t), indicating that taking actions from the states depends on the policy of MCTS-improved. The learner randomly samples and updates the best model parameters from the generator's data. In our algorithm, the MCTS improvement policy becomes an online implementation. Given an unknown graph instance, we feed the heuristics, policies, and $\widetilde{Q}$ values obtained from offline self-play training to the online neural network and then apply MCTS to promote the online policy to form the best model. The online search process is demonstrated in Algorithm 2.

4 Experiments

4.1 Experimental setup

CH-Zero does not require expert experience to learn heuristics through self-play, and data generators can generate data. This paper focuses on path optimization problems, and we can generate VRP or TSP instances to train the model as in previous works [38]. Our method is based on Monte Carlo tree search (MCTS) and reinforcement learning (RL), which theoretically makes it more generalization than supervised learning (which relies on label data) and has more search space than other heuristic search methods. Thereby, we can also generate ER and Barabasus–Albert (BA) graphs to extend the model to other similar combinatorial optimization problems on a larger scale [43, 46].

We chose some classic heuristics and the most recent ones based on learning regarding baseline selection. Previous methods based on AlphaGo Zero mostly solved

Algorithm 2 Online Monte Carlo Tree Search

```
Input: Graph instance $G$, Q-value estimator $\tilde{Q}$, objective function $f$, root (initial) node (state) $s_0$, MCTS search number $E$, Maximum time steps $T_{max}$
Output: Feasible solution sequence
1.  for episode $= 1, \dots, E$ do
2.      for $t = 0, \dots, T_{max}$ do
3.          Lookup from the dictionary to obtain $W(s_t, a)$ and $N(s_t, a)$
4.          if $N(s_t, a) == 0$ then
5.              $a_t \leftarrow argmax_a \tilde{Q}(s, \cdot)$
6.          else
7.              Select the action $a_t$ with the maximum PUCT value:
8.              $a_t = argmax_a \left\{ \beta \cdot \frac{\pi_\theta(a|s_t)^\mu \sqrt{\sum_{a'} N(s_t, a')}}{1 + N(s_t, a)} + \frac{W(s_t, a)}{N(s_t, a)} \right\}$
9.              Update $f_\theta$
10.         if $a_t$ is STOP then
11.             Compute estimated value $V_\theta(s_t) = Q(s_t, a_t)$
12.             Add the selected node to the partial tour
13.             Backup along the tour to update visit count $W(s_t, a)$ and $N(s_t, a)$
14.             $W(s_t, a_t) = W(s_t, a_t) + v$
15.             $N(s_t, a_t) = N(s_t, a_t) + 1$
16.             break
17.         end if
18.     end for
19. end for
20. for each tour in the feasible solution set: $s \in \bar{S}$ do
21.     $r_t = f(s_{t+1}) - f(s_t)$
22.     Repeatedly update the model parameters with the DQN algorithm:
23.     $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$ // Equation (18)
24. end for
```

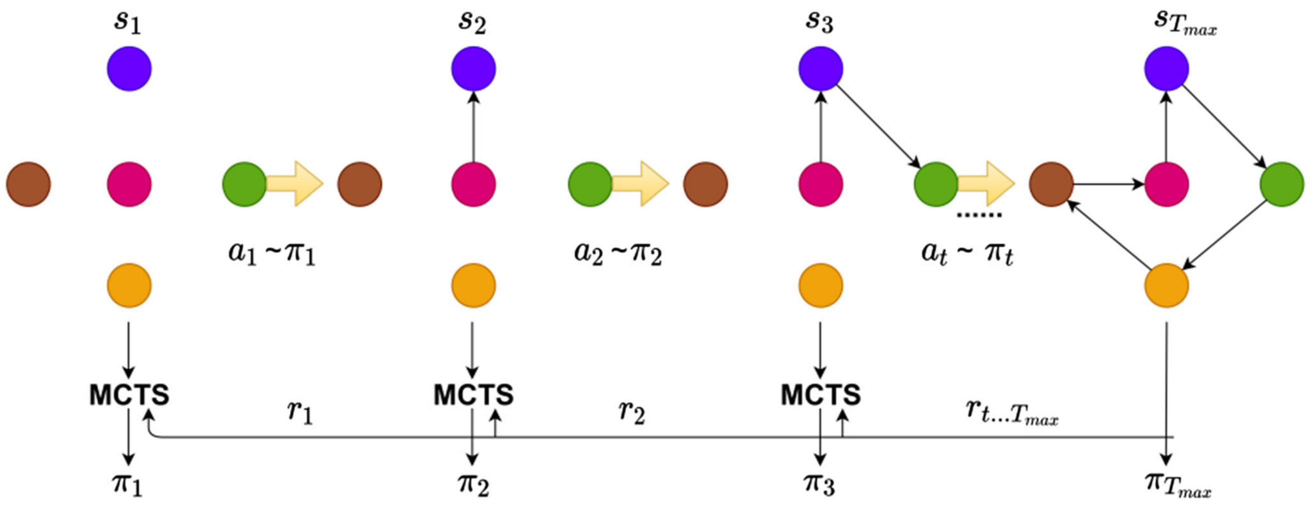

Fig. 3 A simple TSP example is when our method runs online. MCTS calculates the probability of the current move in each state and then selects the following action from samples. The state of the next step and the current state determine the reward for each step

graph coloring, maximal clique (MC), minimum vertex cover (MVC), etc. They did not specifically solve the path optimization problem, so we did not choose them as baselines. Due to the difficulty of reproducing previous works (such as computing resources and time), we chose the experimental tasks and parameter settings similar to previous works as far as possible for convenience comparison. We take the experimental results of baselines in the comparison experiment from their respective papers.

We conduct online and offline evaluations of CH-Zero on the two path optimization problems mentioned. For the GNN setting, we follow [40] and set the dimensions of the three fully connected layers to 256. The hidden dimension of each layer was set to 100, and the number of MLP layers utilized to generate the original features was set to 5. For the pointer recurrent neural network, we set the LSTM hidden dimension to 200 and the attention dimension to 100. Our hyperparameter settings for DQN are as follows: discount rate $\gamma = 1$, batch size B=128, replay buffer size 5 e^3, learning rate 1 e^{-3}, gradient norm clipping 2 e^2 and 1 step reward estimation. For the online MCTS, we employ the same settings as AlphaGo Zero [17]. We set the constant c in Eq. 17 to 1.5, the number of simulations per move of the MCTS to 128, and the temperature parameter for adjusting training to 1. Each time an MCTS instance moves, it runs 128 simulations (for TSP20, TSP50, and TSP100) and 300 simulations (for other instances). Each learner took 20 trajectories from the self-play record and ran a random gradient descent through Adam with a learning rate of 0.001, a weight attenuation of 0.0001, and a batch size of 16. After 15 iterations, the learner saves the new model. Accordingly, we allocated 20, 6 and 2 data generators, learners, and model evaluators. Meanwhile, we use PBT [60] to improve the network's parameters, a population-based strategy for speeding up and enhancing AlphaZero.

Self-play reinforcement learning is notoriously time-consuming, so to thoroughly learn the data distribution and release the model's potential, we give sufficient training time to instances of different sizes. Specifically, for different tasks, we train the model on TSP20 for one hour, TSP50 for two hours, TSP100 and VRP100 for six hours, VRP200, VRP400, and TSP500 for 12 h, and finally Train on TSP1000 for 24 h. We give the longest training time, usually within which learning can converge. The experiments are performed on eight NVIDIA GeForce GTX1080Ti GPUs, and part of the code is available at: https://github.com/Anonymous-author-code/CH-Zero/

4.2 Effect on small-scale TSP instances

We first verify the efficiency of CH-Zero on a relatively simple small-scale TSP instance. On TSP20/50/100 (TSP instances with 20, 50, and 100 nodes, respectively), the solution solvers [38] (such as Concorde, LKH3, and OR-Tools) can already obtain the optimal solution, so we employ the optimal solution as the baseline of the approximate ratio. Specifically, we choose traditional heuristic algorithms [38] such as Nearest, MST, and current state-of-the-art learning-based methods (such as AM [38], GCN [47], S2V-DQN [43], GPN [50]) as baselines. Then we calculate the approximate ratio of the results obtained by different methods to the optimal solution and employ this to measure the accuracy of different methods on small-scale TSP. The closer the approximation ratio is to 1, the closer the result is to the optimal solution. On the contrary, the farther the approximation ratio is from 1, the farther the result is from the optimal solution.

Comparing the results in Fig. 4 demonstrates that CH-Zero is close to the optimal solution on the TSP20/50/100 and surpasses traditional heuristic algorithms and current learning-based methods. These learning-based methods we compared are the same as CH-Zero, which constructs solutions from scratch by decoding them individually. These learning-based methods are all based on the encoder–decoder framework. They learn construction heuristics and data distribution through GNN and reinforcement learning and then decode feasible solutions through heuristic search. CH-Zero differs from them because it learns to improve the heuristics through self-play reinforcement learning. Compared to the beam search and greedy algorithms commonly utilized in previous methods, the policy can be improved iteratively. We think the MCTS in CH-Zero can help exhaust the combination space more precisely. In addition, our unique memory unit for memorizing the path trajectory is also helpful in improving the solution quality because when the agent remembers the path it has traveled, it is beneficial to judge the path to be taken next. There is still a big gap between traditional algorithms based on construction heuristics and solution solvers. The solution solver generally integrates the generation heuristic and the improvement heuristic. The quality of the solution is improved through continuous improvement after the initial solution is obtained. The accuracy of learning-based methods on small-scale instances is much higher than traditional algorithms based on construction heuristics.

4.3 Generalization effects

Generalization is essential for learning-based methods. It includes two aspects: one generalization to similar problems, and the other is generalized to larger-scale problems. The theoretical basis of the generalization of learning-based methods is that they have learned the potential distribution of the problem instance. They can adopt the learned heuristics if they encounter a similar problem distribution. Hence, we demonstrate the efficiency of CH-Zero on larger-scale VRP and TSP instances to prove its generalization. We train and test CH-Zero on VRP instances with 100, 200, and 400 nodes and TSP instances with 500, 750, and 1000 nodes. In this task, we choose representative learning-based methods (such as PRL [36], AM [38], GCN [47], S2V-DQN [43], and GPN [50]) recently utilized to solve VRP or TSP as the baseline.

The experimental results in Tables 1 and 2 show that CH-Zero's results on large-scale VRP and TSP instances are very competitive. Solution solvers like OR-Tools are far less accurate on large-scale instances than small-scale ones and often consume too much time. On the contrary, although learning-based methods require a longer training

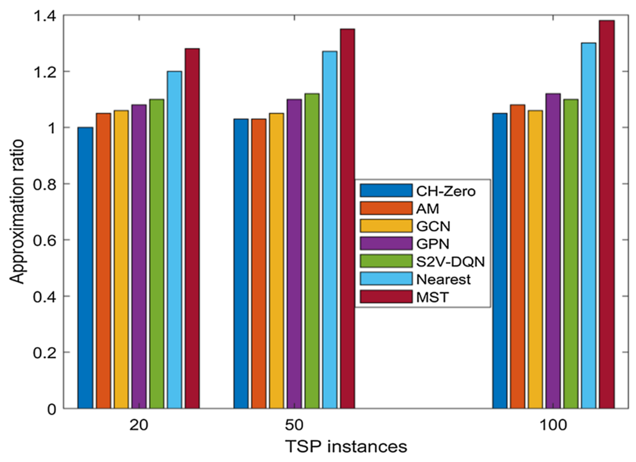

Fig. 4 Comparison of the approximation ratios of different methods to the optimal solution on TSP20/50/100 instances

time than traditional methods, once we complete the training and adopt them for testing, we can complete inference and testing in a short time on unknown instances. Since we did not find the baseline test time on the large-scale VRP in previous works, we only report the time result of CH-Zero here. The test time of CH-Zero on VRP100, VRP200, and VRP400 is 3 s, 5 s, and 8 s, respectively. The training and test sets are the same on TSP experiments, e.g., training on TSP500 is tested on TSP500. Solution solvers have certain advantages in inaccuracy, but the test time consumed is too long and inconvenient in practical applications. More importantly, solution solvers can only target specific problem instances. If we change the structure of problems [such as Eqs. (1)–(5)], then we have to improve the details of the algorithms. Table 1 also found that training on larger-scale instances can get better results. For example, when tested on VRP100, the performance of CH-Zero (200) is better than CH-Zero (100). The performance of CH-Zero (400) is better than CH-Zero (200) when tested on VRP200. Intuitively, the more consistent the training set is with the test set, the better the inference result. But we feel that training on a slightly larger data set for learning-based methods can learn an adequate data distribution, which is more conducive to inference on a slightly smaller data set.

As mentioned earlier, our reward function has a strong generalization, and it can be applied to general path optimization problems with only a slight modification to the Markov decision process. Compared with methods that can only solve a single problem, CH-Zero illustrates desirable universality, a vital aspect of the learning-based method superior to the traditional heuristic algorithm. In terms of modeling, the difference between CH-Zero and other learning-based baselines is that we employ the pointer neural network as the linkage axis so that the value network can drive the policy network and realize parameter sharing to enhance the overall efficiency. In decoding search, we can see that most previous methods employ greedy algorithms or beam search, and the effect of beam search is better. We can conclude that the algorithm based on the heuristic search is better than the simple algorithm based on the nearest neighbor because the search algorithm has a broader field of view. Monte Carlo tree search is also a robust heuristic search algorithm. It can effectively explore large-scale combinatorial space and improve policies through continuous iterative updates. And it can be well integrated into neural networks and reinforcement learning, which is also why AlphaGo Zero chose it.

4.4 Learning effects

The effect of learning is significant for learning-based methods, so we visualize CH-Zero's learning and reasoning process to analyze its efficiency and effectiveness further. We first observe the training of CH-Zero on TSP20. We select the average results of the first six epochs and the average route length obtained by CH-Zero at each time step in each epoch. The learning scatter plots in Fig. 5 show that CH-Zero has a significant gradient descent in the first epoch and converges after the first epoch. After the first epoch, the other epoch curves gradually stabilized and converged toward the optimal solution after the first epoch. Overall, the learning scatter plots on TSP20 are generally relatively stable and can converge quickly, which illustrates that our self-play reinforcement learning algorithm can effectively learn the data distribution on a small-scale TSP.

Then we observe the learning of CH-Zero on a larger-scale routing problem. Specifically, we train CH-Zero for 20 epochs on TSP100 and then obtain the learning curve based on the average route length obtained by CH-Zero after each epoch. The experimental results in Fig. 6 show that the learning curve shows irregular fluctuations, which is reasonable for reinforcement learning. Still, the learning curve fluctuates in a relatively small range. It tends to stabilize and converge as the epoch increases, proving CH-Zero's effectiveness for learning on larger-scale instances. Since we did not join the Monte Carlo tree search in the offline training phase, CH-Zero will not have the problem that the model is too heavy to be efficiently trained.

We employ CH-Zero trained on TSP100 to test a new TSP100 to observe the efficiency and effectiveness of its online reasoning. As mentioned earlier, our model has two stages: offline training and online reasoning. Offline training passes the learned data distribution and heuristics to online reasoning to find routes on new instances. We select the results of the first six epochs of offline inference and the average route length obtained by CH-Zero at each time step. From the experimental results in Fig. 7, we can see that the perturbation of the learning is much more significant than offline training, and there does not seem to be a clear trend of plateauing after the first six epochs. Nevertheless, this phenomenon is reasonable because the combination space to search for MCTS on a large-scale

Table 1 CH-Zero's performance on VRP100/200/400

Method	VRP100		VRP200		VRP400	
	Obj	Gap (%)	Obj	Gap (%)	Obj	Gap (%)
OR-Tools	11.348	0.00	17.995	0.00	26.095	0.00
PRL (G)	12.081	6.56	20.021	11.26	30.197	15.72
AM (G)	11.965	5.44	18.986	5.11	29.589	13.39
GCN (G)	11.347	− 0.01	17.894	− 0.56	27.808	6.56
PRL (BM)	11.907	4.93	19.521	8.48	28.846	10.54
AM (BM)	11.746	3.51	18.697	3.90	27.143	4.02
GCN (BM)	11.276	− 0.06	17.679	− 1.75	27.037	4.02
PRL (BMS)	11.901	4.87	19.019	5.69	28.438	8.98
AM (BMS)	11.643	2.60	18.516	2.90	26.781	2.63
GCN (BMS)	11.259	− 0.08	17.660	− 1.86	26.093	− 0.01
CH-Zero (100)	11.234	− 1.00	17.632	− 2.02	26.176	0.31
CH-Zero (200)	**11.194**	**− 1.36**	17.561	− 2.41	25.964	− 0.50
CH-Zero (400)	11.250	− 0.86	**17.557**	**− 2.60**	**25.883**	**− 0.81**

We utilize the state-of-the-art solution solver OR-Tools as a benchmark to measure the performance of different methods relative to it. Different methods may have different search algorithms, such as greedy algorithm (G), beam search (BM), and beam search with the highest solution quality (BMS). We list all the results of methods with different search algorithms. "Obj." represents the average optimal solution obtained by different methods, and "Gap" represents the gap between optimal solutions

Bold numbers indicate the optimal solution

Table 2 Comparison of model performance on large-scale TSP

Method	TSP500		TSP750		TSP1000	
	Obj	Time (s)	Obj	Time (s)	Obj	Time (s)
Concorde	16.55	13902	**20.10**	32993	**23.11**	47804
LKH3	**16.542**	23070	20.129	36840	23.130	50680
OR-Tools	17.449	5000	22.395	5000	26.477	5000
Nearest insertion	20.791	60	25.219	115	28.973	136
2-opt	18.600	1363	22.668	3296	26.111	6153
Farthest insertion	18.288	160	22.342	454	25.741	945
GPN	18.358	974	22.541	2278	26.129	4410
S2V-DQN	18.428	1508	22.550	3182	26.046	5600
AM	24.789	14	28.281	42	34.055	136
CH-Zero	17.549	**12**	21.157	**27**	25.627	**110**

When some models are combined with heuristics, we choose the best results among different versions

Bold numbers indicate the optimal solution

instance is very large and more complicated than simply learning on one instance.

On the other hand, we adopt the offline trained heuristics and data distribution, but they are not entirely consistent for new instances. Thus, CH-Zero will continue to improve its policies to adapt to new examples during online testing using MCTS. In other words, although the heuristics learned during offline training are obtained in the online reasoning stage, CH-Zero still needs to apply MCTS to continuously explore and learn new heuristics and distributions for self-improvement during the inference process. Offline training is much lighter in the online reasoning phase, leading to stronger robustness. Our application of MCTS to online reasoning has also improved the generalization of CH-Zero from another perspective. Because facing different new instances, we can employ MCTS to enhance the relatively lightweight trained model online to make it more adaptable to new examples, so online reasoning also plays a fine-tuning role. The figure's learning curves corresponding to epoch1, epoch2, and epoch3 are relatively stable, while the following curves are beginning to be unstable. The MCTS gradually plays the role of improving heuristics as the reasoning deepens.

5 Conclusion

This paper proposes a novel self-play reinforcement learning method (CH-Zero) inspired by AlphaGo Zero for mastering construction heuristics. Unlike previous methods based on self-play reinforcement learning, we did not incorporate Monte Carlo tree search (MCTS) into the

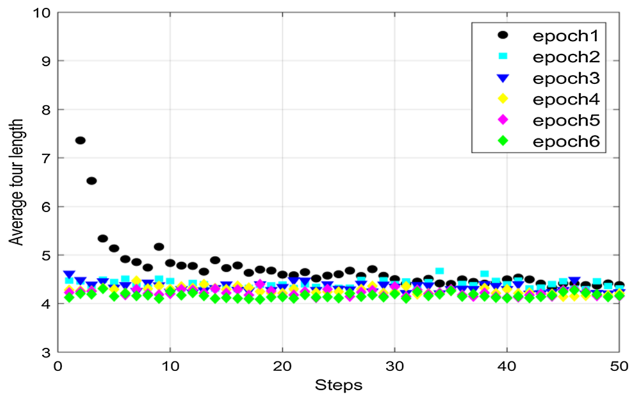

Fig. 5 CH-Zero's learning scatter plots on TSP20. We selected the first six epochs of CH-Zero training on TSP20 and the average route length obtained by CH-Zero at each time step in each epoch (an epoch contains multiple time steps)

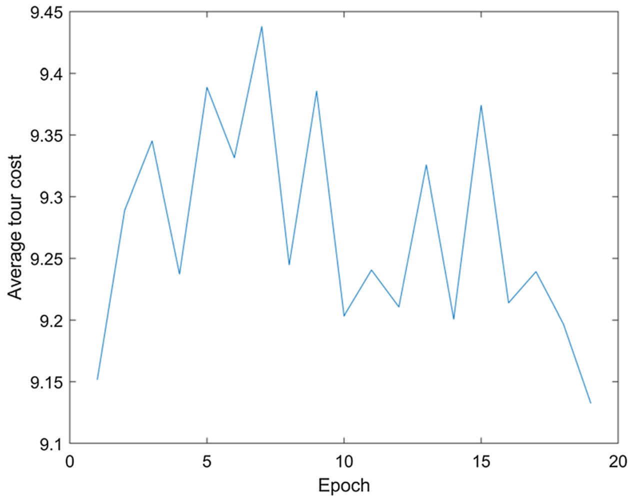

Fig. 6 CH-Zero's learning curve on TSP100. We selected the first 20 epochs of CH-Zero training on TSP100 and the average route length obtained after each epoch

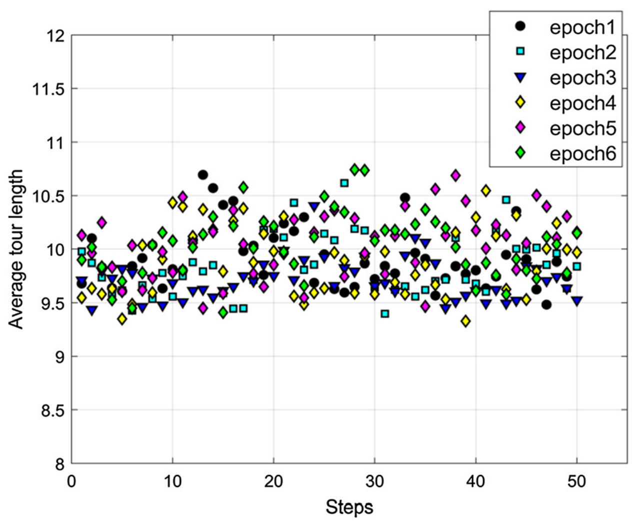

Fig. 7 We employ CH-Zero trained on TSP100 for reasoning on the same data set. We select the results of the first six epochs and the average route length at each time step in each epoch to observe the changes in the results as the time step increases

offline training phase of the model. Still, we incorporated it into the online reasoning phase. The absence of MCTS makes the offline training of the model more lightweight, and it can give full play to the powerful search capabilities of MCTS in online reasoning. We designed a pointer recurrent neural network to construct a parameter-sharing linkage mechanism. The parameters of the value network and the policy network can be updated simultaneously, improving learning efficiency. Therefore, we have contributed to modeling and training, and experiments have shown the efficiency and effectiveness of our method. Continuing to deeply study the core ideas of the AlphaGo series and integrate them into the combinatorial optimization problem on the graph will be our future work.

Declarations

Conflict of interest We wish to confirm no known conflicts of interest associated with this publication. There has been no significant financial support for this work that could have influenced its outcome. We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that all have approved the order of authors listed in our manuscript. We confirm that we have given due consideration to the protection of intellectual property associated with this work. There are no impediments to publication, including the timing of publication, concerning intellectual property. In so doing, we confirm that we have followed the regulations of our institutions concerning intellectual property. We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). They are responsible for communicating with the other authors about progress, submissions of revisions, and final approval of proofs. We confirm that we have provided a current, correct email address accessible by the Corresponding Author and configured to accept email from 17110240039@fudan.edu.cn.

Ethical approval Our study did not raise any ethical questions, i.e., none of the subjects were humans or living individuals. This paper only focuses on combinatorial optimization of graphs in computer science. The technologies used are all modern computer technologies, including deep learning, reinforcement learning, and Monte Carlo tree search. Our research belongs to theoretical and application innovation in computer science, so it does not involve ethical and moral issues.

Informed consent All authors are aware of this article and agree to its submission.

References

1. Prügel-Bennett A, Tayarani-Najaran MH (2012) Maximum satisfiability: anatomy of the fitness landscape for a hard combinatorial optimization problem. IEEE Trans Evol Comput 16:319–338. https://doi.org/10.1109/TEVC.2011.2163638
2. Hernando L, Mendiburu A, Lozano JA (2016) A tunable generator of instances of permutation-based combinatorial optimization problems. IEEE Trans Evol Comput 20:165–179. https://doi.org/10.1109/TEVC.2015.2433680
3. Garey MR, Johnson DS (1979) Computers, Complexity, and Intractability. A Guid. to Theory NPCompleteness, vol 115
4. Xu X, Li J, Zhou MC (2021) Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems. IEEE Trans Intell Transp Syst 22:1583–1593. https://doi.org/10.1109/TITS.2020.2972389
5. Rokbani N, Kumar R, Abraham A, Alimi AM, Long HV, Priyadarshini I, Son LH (2021) Bi-heuristic ant colony optimization-based approaches for traveling salesman problem. Soft Comput 25:3775–3794. https://doi.org/10.1007/s00500-020-05406-5
6. 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
7. Kim G, Ong YS, Heng CK, Tan PS, Zhang NA (2015) City vehicle routing problem (city VRP): a review. IEEE Trans Intell Transp Syst 16:1654–1666. https://doi.org/10.1109/TITS.2015.2395536
8. Goyal S (2010) A survey on travelling salesman problem. In: Midwest instruction and computing symposium, pp 1–9
9. Arnold F, Gendreau M, Sörensen K (2019) Efficiently solving very large-scale routing problems. Comput Oper Res 107:32–42. https://doi.org/10.1016/j.cor.2019.03.006
10. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444. https://doi.org/10.1038/nature14539
11. 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
12. Wang Q, Tang C (2021) Deep reinforcement learning for transportation network combinatorial optimization: a survey. Knowl-Based Syst. https://doi.org/10.1016/j.knosys.2021.107526
13. Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games 4:1–43. https://doi.org/10.1109/TCIAIG.2012.2186810
14. 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
15. 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
16. Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T, Lillicrap T, Silver D (2020) Mastering Atari, Go, chess and shogi by planning with a learned model. Nature 588:604–609. https://doi.org/10.1038/s41586-020-03051-4
17. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, Van Den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of Go without human knowledge. Nature 550:354–359. https://doi.org/10.1038/nature24270
18. 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
19. Huang Y (2020) Deep Q-networks. Deep Reinf Learn Fundam Res Appl. https://doi.org/10.1007/978-981-15-4095-0_4
20. 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
21. Wang Q (2021) VARL: a variational autoencoder-based reinforcement learning Framework for vehicle routing problems. Appl Intell. https://doi.org/10.1007/s10489-021-02920-3
22. Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur J Oper Res 290:405–421. https://doi.org/10.1016/j.ejor.2020.07.063
23. Lodi A, Zarpellon G (2017) On learning and branching: a survey. TOP 25:207–236236. https://doi.org/10.1007/s11750-017-0451-6
24. Toth P, Vigo D (2002) Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discret Appl Math 123:487–512. https://doi.org/10.1016/S0166-218X(01)00351-1
25. Gasse M, Chételat D, Ferroni N, Charlin L, Lodi A (2019) Exact combinatorial optimization with graph convolutional neural networks
26. Ene A, Nagarajan V, Saket R (20180 Approximation algorithms for stochastic k-TSP. In: Leibniz International Proceedings in Informatics, LIPIcs, vol 93, pp 1–11. https://doi.org/10.4230/LIPIcs.FSTTCS.2017.27
27. Sato R, Yamada M, Kashima H (2019) Approximation ratios of graph neural networks for combinatorial problems. Adv Neural Inf Process Syst 32:1–15
28. Sheldon F, Cicotti P, Traversa FL, Di Ventra M (2020) Stress-testing memcomputing on hard combinatorial optimization problems. IEEE Trans Neural Netw Learn Syst 31:2222–2226. https://doi.org/10.1109/TNNLS.2019.2927480
29. Kumar SN, Panneerselvam R (2012) A survey on the vehicle routing problem and its variants. Intell Inf Manag 04:66–74. https://doi.org/10.4236/iim.2012.43010
30. Helsgaun K (2000) Effective implementation of the Lin–Kernighan traveling salesman heuristic.https://doi.org/10.1016/S0377-2217(99)00284-2
31. Zheng J, He K, Zhou J, Jin Y, Li C-M (2020) Combining reinforcement learning with Lin–Kernighan–Helsgaun algorithm for the traveling salesman problem. In: Proceedings of the AAAI conference on artificial intelligence
32. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. Adv Neural Inf Process Syst 4:3104–3112
33. Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. Adv Neural Inf Process Syst 28:2692–2700
34. 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
35. Ivanov S, D'yakonov A (2019) Modern deep reinforcement learning algorithms
36. Nazari M, Oroojlooy A, Takáč M, Snyder LV (2018) Reinforcement learning for solving the vehicle routing problem. Adv Neural Inf Process Syst 31:9839–9849
37. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. Adv Neural Inf Process Syst 30:5999–6009
38. 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
39. Veličković P, Casanova A, Liò P, Cucurull G, Romero A, Bengio Y (2018) Graph attention networks. In: 6th International conference on learning representations. ICLR 2018—conference track proceedings, pp 1–12
40. Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, Raposo D, Santoro A, Faulkner R, Gulcehre C, Song F, Ballard A, Gilmer J, Dahl G, Vaswani A, Allen K, Nash C, Langston V, Dyer C, Heess N, Wierstra D, Kohli P, Botvinick M, Vinyals O, Li Y, Pascanu R (2018) Relational inductive biases, deep learning, and graph networks, pp 1–38
41. Xu K, Jegelka S, Hu W, Leskovec J (2019) How powerful are graph neural networks? In: 7th International conference on learning representations. ICLR 2019
42. Cui P, Wang X, Pei J, Zhu W (2019) A survey on network embedding. IEEE Trans Knowl Data Eng 31:833–852. https://doi.org/10.1109/TKDE.2018.2849727
43. Dai H, Khalil EB, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. Adv Neural Inf Process Syst 30:6349–6359
44. Wu F, Zhang T, de Souza AH, Fifty C, Yu T, Weinberger KQ (2019) Simplifying graph convolutional networks. In: 36th International conference on machine learning. ICML 2019. 2019-June, pp 11884–11894
45. Li Z, Chen Q, Koltun V (2018) Combinatorial optimization with graph convolutional networks and guided tree search. Adv Neural Inf Process Syst 31:539–548
46. Manchanda S, Mittal A, Dhawan A, Medya S, Ranu S, Singh A (2019) Learning heuristics over large graphs via deep reinforcement learning. Assoc Adv Artif Intell
47. Joshi CK, Laurent T, Bresson X (2019) An efficient graph convolutional network technique for the travelling salesman problem, pp 1–17
48. Drori I, Kharkar A, Sickinger WR, Kates B, Ma Q, Ge S, Dolev E, Dietrich B, Williamson DP, Udell M (2020) Learning to solve combinatorial optimization problems on real-world graphs in linear time. In: Proceedings—19th IEEE international conference on machine learning and applications. ICMLA 2020, pp 19–24. https://doi.org/10.1109/ICMLA51294.2020.00013

49. Duan L, Zhan Y, Hu H, Gong Y, Wei J, Zhang X, Xu Y (2020) Efficiently solving the practical vehicle routing problem: a novel joint learning approach. In: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 3054–3063 https://doi.org/10.1145/3394486.3403356
50. Ma Q, Ge S, He D, Thaker D, Drori I (2019) Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning
51. Lu H, Zhang X, Yang S (2020) A Learning-based iterative method for solving vehicle routing problems. ICLR 3:1–15
52. Huang J, Patwary M, Diamos G (2019) Coloring big graphs with AlphaGoZero
53. 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 (2017) Mastering chess and shogi by self-play with a general reinforcement learning algorithm, pp 1–19
54. Laterre A, Fu Y, Jabri MK, Cohen A-S, Kas D, Hajjar K, Dahl TS, Kerkeni A, Beguir K (2018) Ranked reward: enabling self-play reinforcement learning for combinatorial optimization
55. Abe K, Xu Z, Sato I, Sugiyama M (2019) Solving NP-hard problems on graphs with extended AlphaGo Zero, pp 1–23
56. Zhang Z, Cui P, Zhu W (2020) Deep learning on graphs: a survey. IEEE Trans Knowl Data Eng. https://doi.org/10.1109/tkde.2020.2981333
57. Jin C, Allen-Zhu Z, Bubeck S, Jordan MI (2018) Is Q-learning provably efficient? Adv Neural Inf Process Syst 31:4863–4873
58. Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable MDPs. In: AAAI fall symposium—technical report. FS-15-06, pp 29–37
59. Anthony T, Tian Z, Barber D (2017) Thinking fast and slow with deep learning and tree search. Adv Neural Inf Process Syst 30:5361–5371
60. Wu TR, Wei TH, Wu IC (2020) Accelerating and improving alphazero using population based training. https://doi.org/10.1609/aaai.v34i01.5454