A Deep Reinforcement Learning Based Real-Time Solution Policy for the Traveling Salesman Problem

Zhengxuan Ling, Yu Zhang, and Xi Chen

***Abstract*— The rapid development of logistics and navigation has led to increasing demand for solving route optimization problems in real-time. The traveling salesman problem (TSP) tends to require fast and reliable online solutions, which may not be met by traditional iterative optimization algorithms. In this work, a real-time solution policy is proposed for TSP. The idea is to build a mapping between city information and optimal solutions using deep neural networks. Therefore, when given a new set of city coordinates, the optimal route can be directly and quickly calculated without iteration. Considering the recent advancement in computer vision with deep convolutional neural networks (DCNNs), an image representation is proposed to convert TSP to a computer vision problem. A problem decomposition method is introduced to reduce the mapping complexity. Taking advantage of the powerful fitting capabilities of DCNN, a deep reinforcement learning method is designed without any labeling requirement. The proposed method is superior for real-time applications compared with other algorithms.**

***Index Terms*— Machine learning, deep reinforcement learning, real-time algorithm, deep convolutional neural network, traveling salesman.**

I. Introduction

TRAVELING Salesman Problem (TSP) describes a scenario where a salesman travels to several cities and eventually returns to the depot. The objective of the TSP is to minimize the distance of the traveling while ensuring that each city is visited exactly once [1]. As a typical vehicle routing problem, TSP and its variants are widely applied in many different fields, such as logistics [2] and navigation [3]. In recent years, these fields have been developing rapidly with the increased demand for online services on mobile devices. There is increasing demand for real-time data processing, which places higher requirements on the efficiency of the TSP algorithm.

Manuscript received 13 November 2021; revised 23 May 2022 and 8 October 2022; accepted 20 February 2023. Date of publication 20 March 2023; date of current version 31 May 2023. This work was supported in part by the STI 2030-Major Project 2022ZD0208802; in part by NSFC through the Autonomous Intelligent Unmanned Systems under Grant 62088101; and in part by the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China, under Grant ICT2022B04. The Associate Editor for this article was J. Alvarez. *(Corresponding authors: Yu Zhang; Xi Chen.)*

Zhengxuan Ling is with the College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: zxling@zju.edu.cn).

Yu Zhang and Xi Chen are with the State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: zhangyu80@zju.edu.cn; xi_chen@zju.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TITS.2023.3256563, provided by the authors.

Digital Object Identifier 10.1109/TITS.2023.3256563

Most existing solving algorithms for the TSP are insufficient in real-time demand [4], [5], [6]. These algorithms can be categorized as deterministic or heuristic. Deterministic algorithms, such as exhaustive search, dynamic programming, branch and bound can find the optimal solution. However, due to "combinatorial explosion" [1], it is nearly impossible to find the solution promptly for a large instance. Inspired by biological evolution and physics laws, heuristic algorithms, such as genetic algorithm (GA), ant colony algorithm, simulated annealing algorithm, and artificial neural networks [7], [8] are proposed. All types of algorithms are iterative methods. If the problem parameters, such as the number or location of cities change, their iterative processes need to be re-conducted. Thus, the requirement for real-time solutions can hardly be guaranteed.

The development of deep learning brings new opportunities to solve TSPs with deep neural networks (DNNs). Supervised by a set of problem instances and their optimal results, the trained DNN can predict the solutions for different instances directly without any complex search iteration process; thus, it greatly accelerates the computation speed to meet the real-time application demand. In the presented researches, some DNN-based approaches have been proposed for solving TSP and its variants. Li et al. [9] overviewed the literature for solving vehicle Routing Problems. Vinyals et al. [10] proposed a pointer network (PtrNet) based on recurrent neural network structures. At each step, the output of the PtrNet points to an input city through the self-attention mechanism. Joshi et al. [11] embedded the distance information among cities as a topological graph into the neural networks and designed a graph convolutional network (GCN) to produce the optimized city order. These deep learning-based approaches surpass most heuristic algorithms regarding solution efficiency and stability. However, with increasing city counts, it will be difficult for the supervised training methods to obtain enough training labels.

To eliminate the dependence on label generation, the deep reinforcement learning (DRL) method [12], [13] is introduced. Reinforcement learning (RL) is a technique that enables the algorithms to learn and evolve through interaction with the environment. DRL is the combination of the DNNs and RL. With the powerful fitting capabilities of DNNs, the DRL has been extended to tackle complex dynamic modeling and optimization processes in finance [14], traffic [15], and agriculture [16]. TSP is regarded as a sequential process in the DRL. At each step, the neural network takes one step to decide on the next city according to the current

state. By evaluating the total path length, the RL algorithm increases the probability of action with a short total length. In some literature, the DRL was adopted in training Ptr-Net [17] and GCN [18] for TSP. Compared with supervised learning, the convergence in neural networks' parameters is more challenging in the DRL. To ensure convergence and avoid premature convergence, Kwon et al. [19] proposed the multiple optima (POMO) method to exploit the symmetries in the TSP representation. Sultana et al. [20] proposed an entropy regularized reinforcement learning (ERRL) to provide more stochastic strategies in the training processes. DRL approaches are also applicable to TSP variants. Zhang et al. [21] used policy gradient to solve the problem of dynamic TSP and dynamic pickup and delivery. Based on time-varying data, the solution quality of their method surpasses simulated annealing and dynamic programming algorithms. In [22], TSP with time windows and rejections (TSPTWR) was solved by minimizing customer rejection rate and total path length simultaneously. The valuable information about cities is embedded into the network architecture, and a heuristic algorithm is employed to address the other constraints. These methods above are trained based on GCNs and Recurrent Neural Network (RNN) series network structures. Due to over-smoothing and gradient vanishing/explosion, these structures' performance significantly degrades with increasing neuron layers in the networks.

Compared to these neural network structures, deep convolutional neural networks (DCNNs) are widely studied. Mature structures have been widely applied in object detection [23], semantic segmentation [24], medical [25], et al. Some literature attempted to solve the multiagent path finding (MAPF) problem with DCNNs [26], [27]. The task of the MAPF problem is to find the shortest path for each agent that does not collide with obstacles and other agents. MAPF instances were transformed into "maze-like" images, where pixels were used to represent paths, obstacles, agents, and targets. To train the DCNN, Kulvicius et al. [26] generated numerous shortest paths corresponding to different MAPF instances. The DCNN was expected to output the shape of the shortest paths directly. Liu et al. [27] adopted the DRL to predict the direction of the next step for the agent. The image representation and challenges of solving the TSP and MAPF are quite different in many aspects. In the MAPF, the pixels represent obstacles and roads; in the TSP, there is no obstacle on the road, and the salesman needs to select the city among all the possible choices. The images are used to represent the coordinates of cities and to embed the distance among cities. Also, the pixels used to represent the TSP are relatively sparse among the whole image pixels, while the numbers of pixels for obstacles and paths in the MAPF are balanced. In our previous work [28], an image representation method is proposed to address the TSP, in which the city location information is used to generate a fully-connected image as the input, and the image representing the optimal path is used as the output. Powered by supervised learning, a fully convolutional neural network (FCN) structure is adopted to establish an image-to-image mapping. It can output the "optimal-path" image according to the "fully-connected" image. However, the mapping between city coordinates and the optimal path is very complicated. The DCNN can only tackle a dozen of cities since the number of optimal paths grows exponentially with the increase of the scale of the city counts. A TSP decomposition method and a novel dynamic image representation are proposed to address these issues by reducing the mapping complexity. A DRL training method is also designed to improve the DCNN's solution performance without any optimal solution labels. The main contributions of this work can be summarized as follows:

(i) TSP is decomposed into a sequence of sub-problems. A novel image representation is proposed to represent the sub-problems and the actions of the DCNN. The image representation can fully use the DCNN's powerful feature extraction, combination, and fitting capabilities.

(ii) A simple but efficient rule-based filtering mechanism is designed to ensure that the produced actions by the DCNN satisfy the constraints of TSP.

(iii) A visual method is presented to monitor the neural network reinforcement learning process. The network's learning process can be divided into several stages with interpretability.

(iv) The performance of the proposed method is tested on instances with different city coordinates and counts. The DCNN can provide quick, stable, and promising solutions with excellent generalization capability.

Section II introduces the decomposition and image representation of TSP. Section III describes the DCNN-based deep learning model. Section IV introduces the DRL algorithm used to train the DCNN. Section V presents experiments and performance comparisons of the proposed algorithms. Section VI gives concluding remarks.

II. Traveling Salesman Problem Decomposition and Representation

The key to adopting neural networks to solve TSPs is to learn the mapping between problem parameters and the optimal solutions. The complexity of the mapping significantly affects the effectiveness of the neural network. The mapping of a specific TSP with given parameters is generally easy to learn by a neural network. However, when the parameters of TSP vary, the neural network needs to be re-trained, which will be time-consuming and thus meaningless for real-time applications. To immediately obtain the solution without re-training, one needs to establish a complete mapping from the TSP parameter space to the solution space. Compared to the mapping of a specific TSP, the complexity of this complete mapping is greatly increased. Our previous work [28] successfully fitted a mapping for the ten-city TSP. However, it is still challenging to fit the mapping for TSP with more cities since the mapping complexity grows rapidly with increasing the city number. This work introduces a TSP decomposition method to reduce mapping complexity by proposing a new image representation to transform the decomposed TSP into image classification problems.

A. Mapping of the Traveling Salesman Problem

The mathematical model of the TSP with N cities can be defined as (1):

$$\min Z = \sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} x_{ij}$$

$$s.t.\ \begin{cases} \sum_{j=1}^{N} x_{ij} = 1, & i \in V \\ \sum_{i=1}^{N} x_{ij} = 1, & j \in V \\ \sum_{i \in S} \sum_{j \in S} x_{ij} \le |S| - 1, & \forall S \subset V,\ 2 \le |S| \le N-1 \\ x_{ij} \in \{0, 1\} \end{cases} \tag{1}$$

where d_{ij} denotes the distance of the path between city i and city j, and the discrete variable x_{ij} represents whether the path is passed; V is the set of all the cities, and S is a subset of V. In the objective function, the total distance of the paths is minimized. The first two constraints indicate that each city can only have one path in and out. The third constraint guarantees that no sub-loop exists in the final path loop. Based on (1), the mapping f between the problem parameters and solutions is shown as:

$$\begin{cases} O_N = f\left(D_N\right) \\ f : R^{N \times N} \to I^N \end{cases} \tag{2}$$

where D_N is the distance matrix composed of all d_{ij}. $O_N = \{o_1, o_2, \ldots, o_N\}$ is the index of cities in the optimal solution. For all TSP samples with N cities, D_N spans a continuous parameter space $R^{N \times N}$ with $N \times N$ dimensions and O_N spans a discrete solution space I^N with N dimensions. If the mapping f can be fitted and replaced by a neural network, the neural network is capable of predicting the solution for TSP samples with different D_N only through forward propagation. Since re-iteration is avoided for solving new samples, the solution is greatly accelerated.

The previous work [28] demonstrated that the f of a ten-city TSP can be replaced by the FCN. A large number of ten-city TSP samples with uniformly distributed random city coordinates are generated to train the FCN. The coordinates of the cities are marked on an image, and all possible paths among cities are connected to form a fully-connected image, as shown in Fig. 1. The information of the optimal path is obtained by the deterministic algorithm and represented as an optimal-path image. Through the image representation, the mapping is transformed into an image semantic segmentation problem, and the FCN is trained to segment the optimal path from all possible paths. The experimental results demonstrated that the FCN could generate high-quality results for different TSP samples. However, it is limited to cases with only a dozen of cities. When the city number further increases, the performance of the FCN becomes deteriorating because the rapid growth of the complexity of f exceeds the fitting capability of FCN. The solution space of the TSP is the set of all feasible paths. When the number of cities increases from N to $N+1$, $(N-1) \times (N-1)!$ possible paths are added to the solution space. The dimension of the parameter space also increases by $2N+1$.

B. Problem Decomposition

When the city number increases, it is not ideal to directly fit and replace f with another FCN. Since TSP is a dynamic problem composed of multiple repeated sub-problems, this issue can be addressed by decomposing the problem. Suppose i denotes the city index where the salesman is currently located

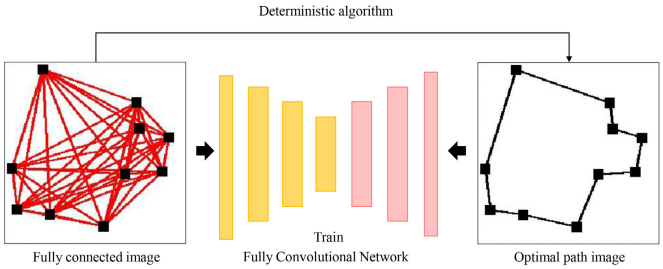


Fig. 1. Solve TSP through the fully convolutional network.

and U_k denotes the set of remaining k cities, then the TSP can be decomposed as:

$$\begin{cases} f_k\left(i, U_k\right) = \min_{j \in U_k}\left(d_{ij} + f_{k-1}(j, U_k \backslash \{j\})\right) \\ f_0\left(i, U_0\right) = f_0\left(i, \emptyset\right) = d_{i1} \end{cases} \tag{3}$$

where $f_k\left(i, U_k\right)$ is the shortest distance of passing all k cities in U_k from i and returning to the depot. j is a city in U_k, and d_{ij} is the distance between i and j. Starting from j, $f_{k-1}(j, U_k \backslash \{j\})$ is the shortest distance to visit all the cities in U_k excluding j and return to the depot. The value of $f_k\left(i, U_k\right)$ can be obtained by traversing all the cities in U_k and selecting the j with the smallest sum of $f_{k-1}(j, U_k \backslash \{j\})$ and d_{ij}. When all cities have been passed ($U_0 \equiv \emptyset$), the value of $f_0\left(i, U_0\right)$ is equal to the distance d_{i1} between city i and the depot. Through (3), the TSP sample with N city can be decomposed into a set of sub-problems with a sub-mapping $f_{sub}^k(\cdot)$. In each sub-problem, the traveling salesman only needs to choose one city from the remaining cities as the next city to visit. Equation (4) presents the mapping $f_{sub}^k(\cdot)$ of each sub-problem:

$$\begin{cases} o_{k+1} = f_{sub}^k\left(D_{m_k}\right), \quad k = 0, 1, 2, \ldots, N-1 \\ f_{sub}^k : R^{m_k \times m_k} \to I^1 \end{cases} \tag{4}$$

where m_k is the total number of cities left in the k^{th} step or the k^{th} subproblem, D_{m_k} denotes the distance matrix of the remaining cities in the subproblem and is equal to D_N when $k = 0$, and o_{k+1} is the index of the selected next city. Meanwhile, the index set of the left cities can be updated to generate the distance matrix in the new subproblem as:

$$D_{m_{k+1}} = g_k(o_{k+1}, D_{m_k}), \quad k = 0, 1, 2, \ldots, N-1 \tag{5}$$

where $g_k(\cdot)$ enables the visited cities to be excluded from D_{m_k}. To solve the TSP by recursively calling $f_{sub}^k(\cdot)$ and $g_k(\cdot)$, the city order can be obtained by:

$$\underbrace{g_{n-1}(f_{sub}^{N-1}(\ldots g_1(f_{sub}^1(g_0(f_{sub}^0(D_N))))))}_{N} \tag{6}$$

Since the nesting number increases with the number of cities, the impact of increasing city number on $f_{sub}^k(\cdot)$ is obviously weakened. The TSP decomposition greatly reduces the mapping complexity, and the neural network can deal with much more cities. After the decomposition, the dimension of the parameter space reduces from N^2 to m_k^2 ($m_k \le N$) in the k^{th} step and the number of possible solutions in the solution space drops from $(N-1)!$ to $k-1$. Taking a TSP sample with 50 cities, for instance, if the final result is given directly

through the neural network, the neural network needs to select a path from 6.08×10^{62} possible paths. After decomposing into sub-problems, the neural network only needs to select one city from 49 cities at most. According to (3), TSP can be considered as a Markov Process since the selection of city j and the generation of subsequent sub-problems solely depend on $f_k(i, U_k)$ (non-aftereffect property). Solving these sub-problems can guarantee the global optimal solution in theory through RL [29]. After inputting the D_{m_k} to the DCNN, the information of the next city for training the neural network can be obtained through the Monte Carlo method [30] (see Section IV for detail).

C. Image Representation of the Sub-problems

After the decomposition, a DCNN is adopted to fit $f_{sub}^{k}(\cdot)$. The information contained in D_{m_k} should also be contained in the input of the neural network, and the neural network can produce the index of the next. D_{m_k} are replaced by the city coordinates by calculating the Euclidean distance among cities because the city coordinates are a more concise representation of the problem. These city coordinates are accommodated in images. Each city coordinate is represented by a pixel in the image. Through image representation, the sub-problem is transformed into an image classification problem. The neural network classifies each pixel in the image and determines which pixel belongs to the next city. Taking a TSP sample with six cities, for example, Fig. 2 shows the image representation of its sub-problems in 6×6 images. For the input image of $f_{sub}^{0}(\cdot)$, white pixels (value=0) represent the background, red pixels (value=-1) represent the coordinates of the current city, and black pixels (value=1) represent the coordinates of the remaining cities. In the output image of the neural network, the pixel of the next city selected is marked in red. Then the selected next city becomes the current city in the $f_{sub}^{1}(\cdot)$. In the input image, the cities passed previously are cleared into the background while the location of the depot is marked in blue (value=2).

In the image representation, fixed-size images are adopted to represent TSP samples with different numerical scales. The coordinates of the cities need to be normalized to locate them accurately in the images. Suppose (c_x^i, c_y^i) denotes the coordinates of the city i, and the location in the image is denoted as (p_x^i, p_y^i). Then the function between (c_x^i, c_y^i) and (p_x^i, p_y^i) is:

$$\begin{cases} p_x^i = \left| (c_x^i - c_x^{min} \Big/ c_{x\&y}^{max}) \times S_{img} \right| \\ p_y^i = \left| (c_y^i - c_y^{min} \Big/ c_{x\&y}^{max}) \times S_{img} \right| \end{cases} \tag{7}$$

where c_x^{min}, c_y^{min} are the minimum values among all the horizontal or vertical city coordinates. $c_{x\&y}^{max}$ is the maximum value among both city coordinates after subtraction. S_{img} is the size of the image, which is set at 40 in this work. $|\cdot|$ is the rounding function.

III. Deep Learning Model Architecture

After dividing TSP into multiple sub-problems and representing them with images, we can build a DCNN to implement the selection of the next city in each subproblem. The architecture of the DCNN is shown in Fig. 3. First, the DCNN receives an input image containing the current city and the remaining cities. Subsequently, operations such as convolution and max-pooling are performed repeatedly on the input image to generate feature maps. After going through an average pooling, a fully connected layer, and a SoftMax operation, the DCNN outputs a probability value of each pixel being the next city. Besides, an effective filtering mechanism is designed to ensure that the results meet the constraints in (1). The details are described as follows.

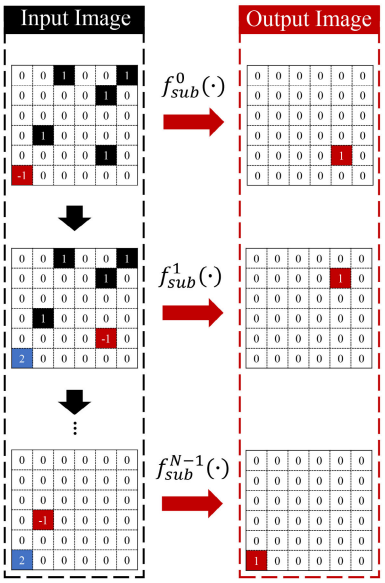


Fig. 2. Image representation of the sub-problems.

A. Deep Convolutional Neural Network

DCNNs have extraordinary capabilities in extracting and combining image features. In recent years, many important researches on DCNN, such as Alexnet [31], VGG [32], and Resnet [33], have been proposed and greatly improved the performance of neural networks in image tasks. These network structures can transform shallow image features (location of points) into more abstract and global features (the positional relationship among cities) through multiple convolutions and pooling operations. In this work, the 16 neural layer version of VGG is adopted as the basis of the DCNN. Some hyper-parameters of the VGG are fine-tuned to meet the needs of the task. After the DCNN receives the 40×40 sub-problem image, the DCNN first performs two convolution operations. The convolution kernel size is 3×3, and the convolution kernel number is 64. To alleviate gradient dispersion and speed up the convergence, batch normalization [34] is performed after convolutions. After that, a $20 \times 20 \times 64$ feature map is generated through the max-pooling down-sampling operation. Based on the generated feature map, the operation combination consists of convolution, batch normalization, and max pooling is performed repeatedly. After each operation combination, the feature map's length and width are halved. The number of convolution kernels is sequentially increased from 64 to 128,

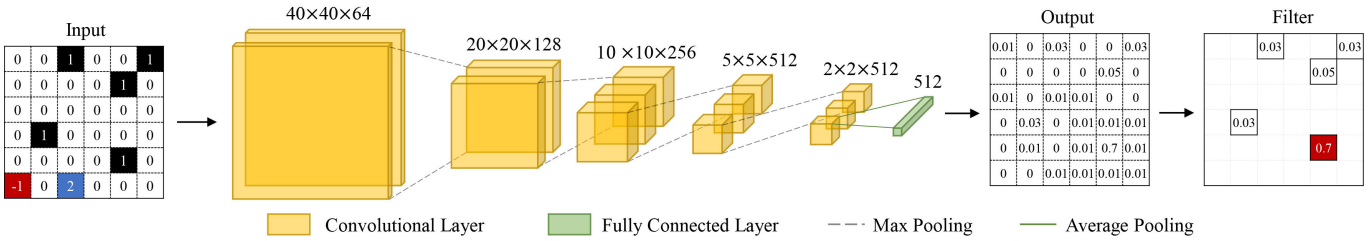


Fig. 3. The structure of the DCNN and filtering mechanism.

256, 512, 512 to compensate for the feature loss due to the reduced size of the feature map. Thus, after five operation combinations, the input image becomes a $2 \times 2 \times 512$ feature map. Subsequently, through average pooling, each channel of the feature map is averaged and used to generate a 512-neuron layer. Then, this neuron layer is fully connected to a 1600-neuron layer (40×40), which is consistent with the size of the input image. Finally, the 1600-dimension neuron values are converted into probability values through the SoftMax operation. Each probability value denotes the probability of a pixel being selected as the next city.

B. Filtering Mechanism

Use set P to store the output probability value of each pixel predicted by the DCNN. Normally, all the probability values in P could be non-zero, indicating that all pixels have a chance to be selected as the next city. Unlike the usual way in deep learning to select the one with the highest possibility, a filtering mechanism with background knowledge is proposed. The constraints in (1) require that a feasible solution of the TSP should satisfy:

(i) Each city should be passed once and only once.

(ii) In the end, the salesman must return to the depot.

The pixels in the input image can be divided into five categories, background pixels, passed city pixels, remaining city pixels, current city pixels, and depot pixels. If a background pixel or a passed city pixel is outputted to have the highest possibility, it cannot be selected to avoid infeasible solutions.

Thus, to satisfy condition (i), the probability values in background pixels and passed city pixels are set to 0. To satisfy condition (ii), the probability value of the depot pixel is also set to 0 except on the last step.

IV. Reinforcement Learning Algorithm

In supervised learning, a large number of labels are required for training the DCNN. It is time-consuming to run different TSPs with traditional algorithms to obtain optimal solutions and generate labels. With RL, however, the tedious label generation process is no longer needed. DCNN can learn from the solving policies generated by itself. A parallel sampling method is designed to address the time-consuming problem during the policy generation and evaluation period. Also, a visualization method is presented to capture the formation of the DCNN mapping during the reinforcement learning process.

A. Background Knowledge of Reinforcement Learning

RL is a process in which an agent repeatedly interacts with the environment and improves its performance. During the interaction, the agent takes an action a based on the current state s. Then, the environment feeds back an immediate reward $r(s, a)$ according to the action and generates a new state. Given states, a policy π_θ is a distribution over actions, decided completely by a DNN with parameter θ. For an interaction that has N states, the probability of following a certain action\state trajectory τ under the policy π_θ is:

$$p_\theta(\tau) = \prod_{t=1}^{N} p_\theta(a_t|s_t) \tag{8}$$

and the expected reward of taking τ is:

$$R_\theta = \sum\nolimits_\tau R(\tau) p_\theta(\tau) \tag{9}$$

where $R(\tau)$ is the cumulative reward of the trajectory. The goal of RL is to maximize the expected reward by optimizing θ, which is usually updated by the gradient descent method. Thus, it is necessary to solve the differential of R_θ with respect to θ as:

$$\begin{aligned} \nabla R_\theta &= \sum\nolimits_\tau R(\tau)\, p_\theta(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)} \\ &= \sum\nolimits_\tau R(\tau)\, p_\theta(\tau) \nabla \log p_\theta(\tau) \end{aligned} \tag{10}$$

B. Deep Reinforcement Learning Algorithm for the Traveling Salesman Problem

In a TSP, the agent is the salesman, and the environment is the TSP sample with given city coordinates. In Section III, a TSP sample with n cities can be decomposed and solved with n steps. The input image representation of each step is a state, and the selection of the next city is an action. When the next city is selected, the immediate reward $r(s, a)$ is:

$$r(s, a) = -d_{ij} \tag{11}$$

where d_{ij} is the Euclidean distance between the selected city and the current city. When calculating the differential of expected reward, it is nearly impossible to traverse all trajectories. Thus, the Monte Carlo method [29] is adopted to select an action randomly under a specific state. Specifically, a random number between [0,1) is generated. The action will be selected when the number falls within the interval of a specific action generated according to the action probability. If K TSPs are sampled using the Monte Carlo method, then ∇R_θ approximately equals to:

$$\nabla R_\theta = \sum\nolimits_\tau R(\tau)\, p_\theta(\tau) \nabla \log p_\theta(\tau)$$

$$\approx \frac{1}{K}\sum_{k=1}^{K}\sum_{t=1}^{N} R(\tau^k)\nabla \log p_\theta(a_t^k|s_t^k) \quad (12)$$

where the cumulative reward $R(\tau_t^k)$ can be calculated as:

$$R(\tau_t^k) = \sum_{t'=t}^{N} \gamma^{t'-t} r(s_{t'}^k, a_{t'}^k) \quad (13)$$

where γ is the discount coefficient as a compromise between solution optimality and learning efficiency. When γ is smaller than 1, the reinforcement learning policy pays more attention to the selection of the next few cities and can effectively speed up the convergence of neural networks. However, if a very small discount factor is chosen, the reinforcement learning policy becomes a greedy search. In this work, the discount factor is set to 0.99, considering the quality of the solutions and hardware conditions. It should be noted that all the $R\left(\tau_t^k\right)$ are negative and different in values. When a state/action pair is sampled, the neural network will be trained to reduce the probability of this state/action pair next time. Equation (13) works well when the sampling is uniform and sufficient since the probability of state/action pairs with larger absolute values decreases faster. However, this may lead to unreasonable probability reductions for sampled pairs when the sampling is insufficient. Thus, $R\left(\tau_t^k\right)$ needs to be further processed as:

$$A(\tau_t^k) = (R(\tau_t^k) - R(\bar{\tau}_t^k))/R(\bar{\tau}_t^k) \quad (14)$$

and ∇R_θ becomes:

$$\nabla R_\theta \approx \frac{1}{K}\sum_{k=1}^{K}\sum_{t=1}^{N} A(\tau_t^k)\nabla \log p_\theta(a_t^k|s_t^k) \quad (15)$$

Both $R(\tau)$ and $R(\bar{\tau})$ are obtained under π_θ following different trajectory sampling methods. The actions in τ are selected through the Monte Carlo method, while the ones in $\bar{\tau}$ are the actions with the highest probability value [35]. Through (14), $A(\tau^m)$ is normalized to the same numerical scale, which is conducive to training convergence [36].

In this work, 20,000 TSP training samples with 50 cities in each sample are generated. The coordinates of the cities are randomly and evenly distributed. The DRL process is performed on the whole training sample set to generate policies. To test the DCNN's capability to solve new TSP samples, an additional set of 200 TSP samples were generated as testing data. The DRL algorithm for the TSP is shown in Table I in the form of the pseudo-code that can be divided into two parts: sampling and optimization. The sampling process is executed multiple times on different TSP samples randomly selected from the training data to obtain sufficient data for $\{s, a, A(\tau)\}$. When optimizing the DCNN parameters θ, M data pairs of $\{s, a, A(\tau)\}$ are randomly selected to form a training batch. The random selection ensures that TSP samples with different city coordinates have an equal chance of being encountered by the DCNN.

C. Parallel Sampling Method

Sampling is a time-consuming process that can significantly slow down the DRL process. For a TSP with N cities, the DCNN is repeatedly called $N(N+3)/2$ times to obtain N sets of $\{s, a, A(\tau)\}$ samples. Most of the time, the DCNN performs forward propagation to fetch policies, occupying only a small portion of computing resources.

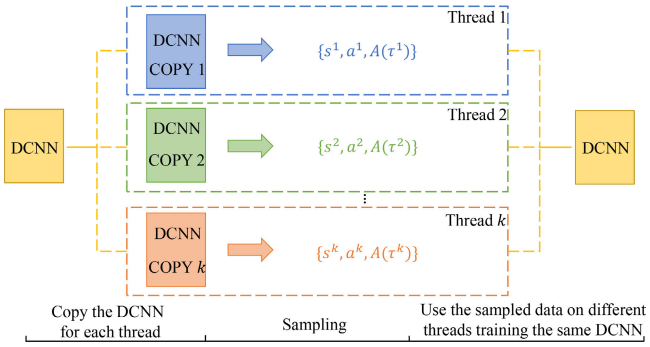


Fig. 4. Parallel sampling process.

Executing the sampling process in parallel is an effective method [37]. For a single sampling process, the subsequent state strongly depends on the previous state and is thus difficult to parallelize. However, the sampling processes on different TSPs are completely independent. The parallel sampling process on different threads is illustrated in Fig. 4. The decomposition and synchronization of the tasks are implemented in Pytorch with the following procedures.

(i) Randomly select k sets of TSP samples from the training data; copy the DCNN to k copies; place a TSP sample and a DCNN copy in a thread.

(ii) Each replicated DCNN with a TSP sample generates $\{s, a, A(\tau)\}$ on a thread independently.

(iii) Randomly select samples and train the DCNN. When the parameters of the DCNN are updated, the parameters of the copied DCNNs are also updated simultaneously.

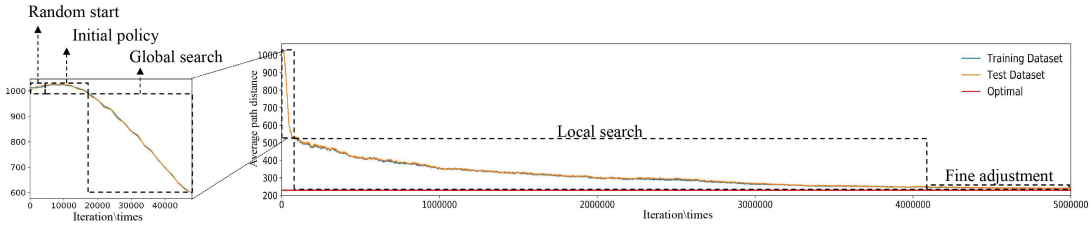
Table II shows the time consumption of 2000 sets of TSP samples under the different number of threads. When the thread number is increased from 1 to 8, the sampling time is accelerated by 3.5 times. Ideally, if k threads are adopted for the parallel sampling, the sampling time can be accelerated by k times. In actual applications, the acceleration effect is affected by the computing capacity of the single thread and communication among threads.

D. Visualization and Interpretability of Training Process

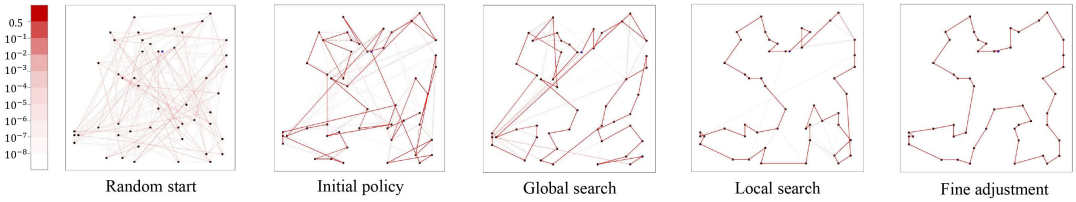
During the DRL process, 32 pairs of $\{s, a, A(\tau)\}$ data are set as a training batch to train the DCNN once. For every 2000 training iterations, DRL algorithms re-sample 500 sets of TSP samples. ADAM [38] is adopted as the optimization algorithm for the loss function, and the learning rate is set to $\alpha = 3 \times 10^{-5}$. The training process lasts 5 million iterations and takes about 280 hours on the PC. The average path distance generated by the DCNN on the training data and the testing data is recorded during the training. As shown in Fig. 5(a), the average distance changes with the iterations to form the learning curve of the DCNN. The blue line and orange line represent the training data and test data, respectively. The red horizontal line shows the average result of the optimal paths on the test set solved by CPLEX, a successful traditional algorithm for TSP.

TABLE I
DEEP REINFORCEMENT LEARNING ALGORITHM

	Sampling
1:	select a TSP sample randomly
2:	for t in range (1, number of cities N):
3:	generates state s_t
4:	DCNN generates the probability of actions
5:	choose action a_t through **Monte Carlo methods** and get immediate reward $r(s_t, a_t)$
6:	end
7:	for t in range (1, N):
8:	calculate $R(\tau_t) = \sum_{t'=t}^{N} \gamma^{t'-t} r(s_{t'}, a_{t'})$
9:	get $R(\bar{\tau}_t)$ through **choosing the action with the largest probability**
10:	calculate $A(\tau_t)$
11:	store $\{s_t, a_t, A(\tau_t)\}$ pair
12:	end
	Optimization
13:	select M pairs of $\{s_t, a_t, A(\tau_t)\}$ randomly
14:	calculate ∇R_θ
15:	back-propagate to optimize θ



(a) Learning Curve



(b) Training Process Visualization

Fig. 5. The deep reinforcement learning process of the DCNN.

TABLE II
THE TIME CONSUMPTION OF SAMPLING

Threads Number	1	2	4	8
Time Consumption/s	114.90	78.43	42.71	33.15

To understand the DRL process of the DCNN more intuitively, a visualization method is designed to monitor the whole training process. Specifically, after every 2000 iterations, three cities with the highest probability of being selected as the next city at each step are recorded. As shown in Fig.5(b), the path between the current city and the city with the highest probability of being next selected is marked by a solid line, and the paths to the other two cities are marked by dotted lines. As revealed by the color bar on the left-hand of Fig.5(b), the higher the probability of a connection, the darker the color. The lines generated in all steps are drawn on the images as shown in Fig. 5(b). These images generated during the whole DRL process have also been made as animations and attached to the manuscript as supplementary files. From the visualization method, the DRL process can be divided into several stages with inherent features.

1) Random Start: At the beginning of the training, the neural network does not have any prior knowledge. All cities have a similar probability of being selected, which can be revealed by the almost same color brightness of the lines.

2) Initial Policy: For a period of iterations after the start, the distance of the path generated by the neural network does not decrease, but the path is no longer selected randomly. From the visualization, one route has a darker color than the others. The DCNN begins to prefer certain paths, though the solution is yet to be optimized.

3) Global Search: The DCNN realizes that the current policy is "clumsy" and begins to search in the global scope of the image. At this stage, two cities very far from each other may also be connected. The policy generated by the DCNN is advancing rapidly.

4) Local Search: At this stage, the DCNN is quite certain about its global policy. The general shape of the path loop has been fixed. Connections among relatively close cities are still alternating, and the generated path distance is decreasing slowly. The other paths, in light lines, now have very low probabilities.

5) Fine Adjustment: After the local search, only fine adjustment is required in the last stage. With sufficient search, the probability value of a certain path dominates. The other possible connections almost completely disappear from the graph. The DCNN will be significantly less willing to continue the path search. In many iterations, the DCNN has a considerable probability of outputting the same solution.

V. Experimental Results and Analysis

In this section, TSP samples with different city coordinates and city counts were generated to test the performance and generalization capability of the DCNN. In this work, all experiments are run on a computer equipped with a 9700K Intel CPU, 2080Ti Nvidia GPU, and 32G memory. The training of the DCNN and the follow-up comparison experiments are all carried out using Python 3.6 with Pytorch 1.6.0 for the development of the DCNN and CPLEX Python API 12.8 as the traditional solver for comparison. The number of parallel threads is set to eight.

A. Performance on Testing Data With Same City Count

First, the performance of the developed DCNN is tested on samples with the same city count as the training data, i.e., 50 cities. Two hundred sets of TSP samples with different coordinates from the training data constitute the testing data. During the training process, the distance length output by the DCNN on the test set is also recorded. As shown in Fig. 5(a), no overfitting occurs during the training. The performance of the DCNN in the training and testing data is consistent, indicating that the DCNN has stable performance in handling the TSP samples with different city coordinates.

Several typical samples in the testing data are shown in Fig. 6. The left column is the solutions produced by the DCNN, and the right column is the optimal solutions solved through CPLEX. The dots in the figure represent the cities, while the red lines represent the paths. In some cases, like the sample in subplot (a), the DCNN obtains exactly the optimal solution. In samples (b) and (c), slight errors exist between the DCNN's solutions and the optimal solutions. The path deviation is further analyzed. The black boxes in the figure represent the deviated local paths from the optimal solution, which all occur in small local areas where the length deviation is tiny from the optimal paths. Such small errors are easily overlooked by the DCNN. However, the overall connection structures are very similar to the optimal solutions.

In general, the DCNN can directly predict solutions that are very close to the optimal solution on most samples. In Tables III and IV, the solution quality and time consumption of the DCNN are compared with CPLEX, LKH [39], and GA algorithms on testing data. By setting Gap=0, the optimal solutions obtained through the CPLEX serve as a baseline for

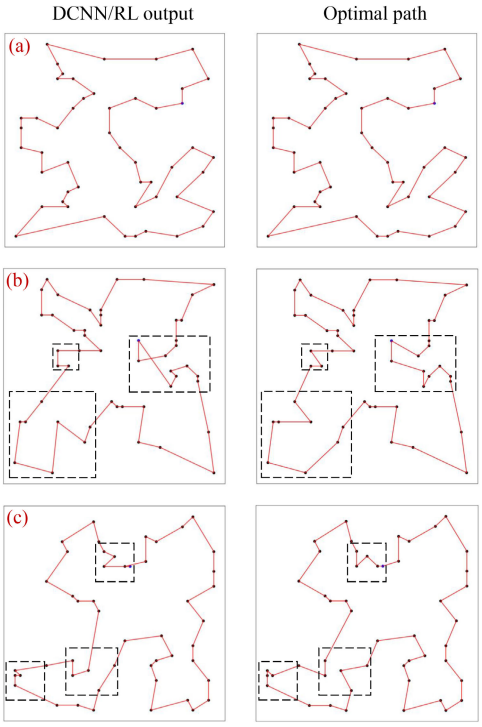


Fig. 6. The solutions of the DCNN on some testing samples.

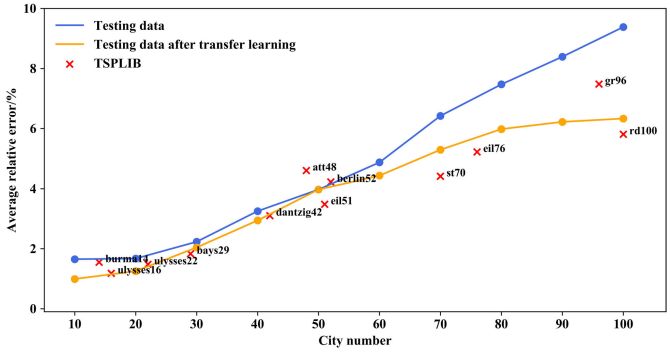


Fig. 7. The performance of the DCNN on different testing data.

accuracy. As shown in the last row of Table III, compared to the baseline, the average relative distance error of the DCNN is only 3.97% for problems with 50 cities. The DCNN requires some time for the training stage. However, once the training is complete, the DCNN can solve new TSP samples in real-time without any further training requirements during the application. On the testing data, the average solution time of the DCNN is only 0.022 s, which is more than 700 times faster than the CPLEX on average. Compared with GA algorithms, the solution speed of the DCNN is over 300 times faster, and the solution quality is much better than the ones of GA. LKH is an advanced heuristic method that is well recognized as one of the best methods for TSP. Compared with LKH, this proposed method still has a significant advantage in solution speed and the time of solving different samples is very stable; meanwhile, the error, though larger than the LKH, is still relatively low. In general, the proposed method has superiority for real-time demands

TABLE III
THE AVERAGE DISTANCE ERROR AND VARIANCE OF SOLUTIONS SOLVED BY DIFFERENT ALGORITHMS

Method	50 Cities		60 Cities		70 Cities		80 Cities	
	Error Aver.	Error Var.	Error Aver.	Error Var.	Error Aver.	Error Var.	Error Aver.	Error Var.
CPLEX	0%	0	0%	0	0%	0	0%	0
LKH	0.39%	1.58×10^{-5}	0.51%	1.41×10^{-5}	0.63%	1.63×10^{-5}	0.73%	2.17×10^{-5}
GA	29.55%	1.10×10^{-2}	42.54%	1.47×10^{-2}	56.28%	1.69×10^{-2}	70.64%	3.76×10^{-1}
DCNN	3.97%	6.49×10^{-4}	4.43%	6.75×10^{-4}	5.29%	7.12×10^{-4}	5.98%	7.93×10^{-4}

TABLE IV
THE AVERAGE TIME CONSUMPTION AND VARIANCE OF DIFFERENT ALGORITHMS' SOLUTION PROCESS

Method	50 Cities		60 Cities		70 Cities		80 Cities	
	Time Aver.	Time Var.	Time Aver.	Time Var.	Time Aver.	Time Var.	Time Aver.	Time Var.
CPLEX	16.53s	2.98×10^{2}	38.39s	1.18×10^{3}	70.56s	1.76×10^{3}	100.10s	2.61×10^{3}
LKH	0.33s	3.72×10^{-3}	0.45s	4.42×10^{-3}	0.52s	8.43×10^{-3}	0.65s	2.13×10^{-2}
GA	7.77s	1.47×10^{2}	11.94s	1.92×10^{2}	15.60s	2.22×10^{2}	23.91s	3.91×10^{2}
DCNN	0.022s	2.15×10^{-5}	0.028s	2.26×10^{-5}	0.038s	2.31×10^{-5}	0.047s	2.52×10^{-5}

Note: Aver. is the abbreviation for average and Var. is the abbreviation for variance

B. Generalization and Transfer Learning for Different City Count

Although trained only for TSP samples with 50 cities, the generalization capability enables the DCNN to solve TSP samples with different city counts. Two hundred additional samples with the city count varying from 10 to 100 are added to test the generalization capability. Without any additional training, the DCNN is directly applied to predict the solutions. Good generalization capability is demonstrated on these testing data sets. As can be seen from the blue line in Fig. 7, when dealing with samples with fewer cities, the relative errors are even smaller than in the 50-city case, indicating even better performance than the training set. The reduction of the solving complexity may contribute to the decrease in relative error. The relative error decrease also indicates that the DCNN has learned the concept of solving TSPs instead of simply recording the image style in the training data. When the number of cities increases, the performance of the DCNN is also controllable. When the city count increases by 10, the relative error only increases by 1%. Therefore, when there are 100 cities, the relative error can be controlled within 10%. Due to the increase in the complexity of the problem, this slow increase in relative error is reasonable.

Besides, transfer learning methods [40] can be applied to further improve the performance of the DCNN for different city counts. Transfer learning is a deep learning technique that adopts learned knowledge to learn similar new knowledge. For the TSP, there are similarities in the solving rules with different city counts. If the network parameters trained for 50 cities are used as initial values, the efficiency of the network learning to solve more cities can be greatly improved. In this experiment, additional 20,000 training sets with a different city count are generated for the transfer learning. Based on the DCNN's parameters trained with 50 cities, another transfer learning process with one million iterations is executed on the training sets with city numbers less than 50. As shown in Fig. 7, the error is further reduced based on the previous accurate results. For TSP samples with more than 50 cities, the transfer learning method is applied step by step. Firstly, the DCNN is trained by training sets in 60 cities. Then, based on the parameters trained for 60 cities, DCNN learns to solve the TSP with 70 cities. This repetitive transfer learning method enables the DCNN to solve TSP samples with larger city counts gradually and fully using the previous network training results. Through this method, the performance of the DCNN is obviously improved at a relatively small training cost.

The performance of the DCNN on TSP testing data with 60, 70, and 80 cities are also compared with CPLEX, GA, and LKH algorithms as presented in Tables III and IV. With the increase in city count, the solution time of the CPLEX and GA algorithms grows significantly. When solving TSPs with 80 cities, this proposed method can generate good-quality solutions in much less time than other algorithms. The comparison results indicate that the DCNN's performance is competitive in solving TSP instances with different city counts.

C. Performance on the Traveling Salesman Problem Benchmark

The TSP samples used in Section V-A and Section V-B are generated with random city coordinates by the authors. In this section, the real city coordinates are adopted to illustrate the effectiveness of the DCNN. TSPLIB [41] is a well-known benchmark collected from real city coordinates that have been widely adopted to test the capability of solving algorithms. The numbers in the name indicate the city count. The performance of the DCNN on different TSPLIB samples is shown as red points in Fig. 7, which are almost evenly distributed along the yellow line. With the increase in the city count, the distribution

TABLE V
THE AVERAGE ERROR OF DIFFERENT DEEP LEARNING ALGORITHMS

	Structure	Training	20 Cities	50 Cities	100 Cities
1	PtrNet [10]	SL	1.15%	34.48%	-
2	FCN [28]	SL	**0.44%**	15.63%	-
3	PtrNet [17]	RL	1.42%	4.46%	6.90%
4	S2V [18]	RL	1.42%	5.16%	7.03%
5	GAT [42]	RL	0.66%	3.98%	8.41%
6	Ours	RL	0.78%	**3.97%**	**6.33%**

Note: SL denotes supervised learning

of the red points is also consistent with the trend and value of the yellow line. Such red dots distribution illustrates that the DCNN also has stable performance in solving real TSP applications.

D. Comparison With Other Deep Learning Methods

With the rapid development of deep learning, some typical deep neural network structures, such as recurrent neural networks and graph neural networks, have also been fine-tuned to solve the TSP. In this section, the performances of these deep learning structures are compared to the method proposed in this work. In addition to DRL, supervised learning is a commonly used method for training neural networks. In our previous work [28], FCN has demonstrated the capability to solve the TSP through supervised learning. In that work, the location of the cities and all possible paths are drawn in the input image. The image semantic segmentation method is adopted by the DCNN to segment the optimal paths from all possible paths directly. The performances of supervised learning and DRL for solving TSPs are also compared in this section.

The third to sixth rows of Table V present different deep learning methods' average error of solutions to TSP problems with 20, 50, and 100 cities, respectively. All these networks are trained using DRL. PtrNet is a type of recurrent neural network; S2V and GAT use graph neural network, which improves the training method and attention mechanism, respectively. These network structures have been widely recognized for excellent TSP solution performance. Compared to these networks, our proposed method has higher solution accuracy in solving 50 and 100 city TSP samples. Moreover, owing to the superior fitting capability of the DCNN, the solution accuracy of this method decreases more slowly with the growth of city count than the other network structures.

The first and second rows of Table V list the performance of PtrNet and FCN training with supervised learning strategy, which requires a large number of labels to be prepared. For a city count as large as 100, it is nearly impractical to generate sufficient optimal solution labels through traditional algorithms. With the advantage of learning directly from optimal solutions, the relative error of the supervised learning for testing data is slightly lower than the DRL with a small city count such as 20 in the test. However, when the number of cities increases, the performance of supervised learning deteriorates significantly, while the performance of DRL is relatively stable. The potential reason lies in: (1) The increase in the number of cities leads to a rapid increase in the complexity of the problem. Through dissembling the TSP step by step, the complexity of the problem in the DRL is significantly reduced. Thus, the mapping to be learned in the DRL is relatively simpler than the mapping to be learned in supervised learning. (2) The goal of the DRL is to directly minimize the length of the path, while the goal of supervised learning is to mimic the features of the labels. Minimizing the length of the path is more consistent with the objective function of the TSP. In general, compared to supervised learning, the DRL is more accurate on more city numbers due to the better generalization performance.

VI. CONCLUSION

In this work, the mapping between TSP and optimal solution is constructed using a DCNN. A problem decomposition method is introduced to reduce mapping complexity. It is extremely time-consuming for traditional algorithms to generate optimal labels when the city count is large, so a deep reinforcement learning method is adopted to train the DCNN. The learning process is accelerated through a parallel sampling method and illustrated intuitively through visualization. The accurate performance and generalization capability of the developed DCNN have been demonstrated through test samples with different city coordinates and city counts. A test on standard TSP benchmarks further illustrates the stable performance of actual TSP applications. The superiority in meeting the real-time demand is also compared with other iterative algorithms. At last, a comparative experiment between deep reinforcement learning and supervised learning is presented, demonstrating that the DCNN trained without any labels achieves better generalization capability than the supervised learning on this problem.

The image representation and solution approach proposed in this work have the potential for applications in other TSP variants. For example, TSP with time windows where some customers are only available for a certain period may be represented the appearance and disappearance of city points in the input images. Moreover, colored TSP [43], [44], [45] may display the depot on and off according to the carrying capacity of each salesman. For multiple TSP, DCNNs can optimize the path for each salesman through multiagent RL techniques [46]. Currently, the scale of the problem that the presented method can solve is still limited by the image size. Designing a unified DCNN-based DRL framework that can handle problems efficiently with different scales and objectives still needs further research.

ACKNOWLEDGMENT

The authors would like to thank Ming Gong with the Bard College, Simon's Rock, for refining the language.

REFERENCES

[1] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *Eur. J. Oper. Res.*, vol. 209, no. 1, pp. 1–10, Feb. 2011, doi: 10.1016/j.ejor.2010.03.045.

[2] S. Tofighi, S. A. Torabi, and S. A. Mansouri, "Humanitarian logistics network design under mixed uncertainty," *Eur. J. Oper. Res.*, vol. 250, no. 1, pp. 239–250, Apr. 2016, doi: 10.1016/j.ejor.2015.08.059.

[3] A. De, J. Wang, and M. K. Tiwari, "Hybridizing basic variable neighborhood search with particle swarm optimization for solving sustainable ship routing and bunker management problem," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 986–997, Mar. 2020, doi: 10.1109/TITS.2019.2900490.

[4] J. N. MacGregor and Y. Chu, "Human performance on the traveling salesman and related problems: A review," *J. Problem Solving*, vol. 3, no. 2, pp. 1–29, Feb. 2011, doi: 10.7771/1932-6246.1090.

[5] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *Eur. J. Oper. Res.*, vol. 255, no. 2, pp. 315–332, 2016, doi: 10.1016/j.ejor.2016.04.059.

[6] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 9, pp. 5057–5069, Sep. 2022, doi: 10.1109/TNNLS.2021.3068828.

[7] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, Jul. 1985, doi: 10.1007/BF00339943.

[8] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59–69, Jan. 1982, doi: 10.1007/BF00337288.

[9] B. Li, G. Wu, Y. He, M. Fan, and W. Pedrycz, "An overview and experimental study of learning-based optimization algorithms for the vehicle routing problem," *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 7, pp. 1115–1138, Jul. 2022, doi: 10.1109/JAS.2022.105677.

[10] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Red Hook, NY, USA: Curran Associates, 2015, pp. 1–9.

[11] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," 2019, *arXiv:1906.01227*.

[12] V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[13] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[14] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017, doi: 10.1109/TNNLS.2016.2522401.

[15] B. R. Kiran et al., "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022, doi: 10.1109/TITS.2021.3054625.

[16] F. Bu and X. Wang, "A smart agriculture IoT system based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 99, pp. 500–507, Oct. 2019, doi: 10.1016/j.future.2019.04.041.

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

[18] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 1–11.

[19] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "POMO: Policy optimization with multiple optima for reinforcement learning," 2020, *arXiv:2010.16011*.

[20] N. Sultana, J. Chan, T. Sarwar, and A. K. Qin, "Learning to optimise routing problems using policy optimisation," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8, doi: 10.1109/IJCNN52387.2021.9534010.

[21] Z. Zhang, H. Liu, M. Zhou, and J. Wang, "Solving dynamic traveling salesman problems with deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 14, 2021, doi: 10.1109/TNNLS.2021.3105905.

[22] R. Zhang, A. Prokhorchuk, and J. Dauwels, "Deep reinforcement learning for traveling salesman problem with time Windows and rejections," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8, doi: 10.1109/IJCNN48605.2020.9207026.

[23] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," 2019, *arXiv:1905.05055*.

[24] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017, doi: 10.1109/TPAMI.2016.2572683.

[25] J. Duan et al., "Automatic 3D bi-ventricular segmentation of cardiac images by a shape-refined multi-task deep learning approach," *IEEE Trans. Med. Imag.*, vol. 38, no. 9, pp. 2151–2164, Sep. 2019, doi: 10.1109/TMI.2019.2894322.

[26] T. Kulvicius, S. Herzog, T. Lüddecke, M. Tamosiunaite, and F. Wörgötter, "One-shot path planning for multi-agent systems using fully convolutional neural network," 2020, *arXiv:2004.00568*.

[27] Z. Liu, B. Chen, H. Zhou, G. Koushik, M. Hebert, and D. Zhao, "MAPPER: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," 2020, *arXiv:2007.15724*.

[28] Z. Ling, X. Tao, Y. Zhang, and X. Chen, "Solving optimization problems through fully convolutional networks: An application to the traveling salesman problem," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 51, no. 12, pp. 7475–7485, Dec. 2021, doi: 10.1109/tsmc.2020.2969317.

[29] C. Archetti, D. Feillet, A. Mor, and M. G. Speranza, "Dynamic traveling salesman problem with stochastic release dates," *Eur. J. Oper. Res.*, vol. 280, no. 3, pp. 832–844, Feb. 2020, doi: 10.1016/j.ejor.2019.07.062.

[30] C.-J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," *IEEE Trans. Intell. Vehicles*, vol. 5, no. 2, pp. 294–305, Jun. 2020, doi: 10.1109/TIV.2019.2955905.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[36] L. Engstrom et al., "Implementation matters in deep policy gradients: A case study on PPO and TRPO," 2020, *arXiv:2005.12729*.

[37] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[38] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–41.

[39] É. D. Taillard and K. Helsgaun, "POPMUSIC for the travelling salesman problem," *Eur. J. Oper. Res.*, vol. 272, no. 2, pp. 420–429, Jan. 2019, doi: 10.1016/j.ejor.2018.06.039.

[40] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2009, doi: 10.1109/TKDE.2009.191.

[41] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, Nov. 1991, doi: 10.1287/ijoc.3.4.376.

[42] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L. M. Rousseau, "Learning heuristics for the TSP by policy gradient," in *Proc. Int. Conf. Integr. Constraint Program., Artif. Intell., Oper. Res.*, 2018, pp. 170–181.

[43] J. Li, M. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2390–2401, Nov. 2015, doi: 10.1109/TCYB.2014.2371918.

[44] J. Li, X. H. Meng, and X. Dai, "Collision-free scheduling of multi-bridge machining systems: A colored traveling salesman problem-based approach," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 139–147, Jan. 2018, doi: 10.1109/JAS.2017.7510415.

[45] X. Xu, J. Li, and M. Zhou, "Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 3, pp. 1583–1593, Mar. 2021, doi: 10.1109/TITS.2020.2972389.

[46] D. Bertsekas, "Multiagent reinforcement learning: Rollout and policy iteration," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 2, pp. 249–272, Feb. 2021, doi: 10.1109/JAS.2021.1003814.

Zhengxuan Ling was born in Jinhua, Zhejiang, China, in 1996. He received the B.S. degree in automation from Central South University, Changsha, in 2018. He is currently pursuing the Ph.D. degree in control science and engineering with Zhejiang University, Hangzhou, China. His research interests include machine learning, optimization algorithms, and computer vision. His awards and honors include National Scholarship and Meritorious Winner of Mathematical Contest in Modeling (MCM).

Yu Zhang was born in Hangzhou, Zhejiang, China, in 1980. He received the B.S. degree in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2003, and the M.S. and Ph.D. degrees in computer science and technology from Tsinghua University, Beijing, China, in 2009. He was a Post-Doctoral Researcher with the State Key Laboratory of Intelligent Technology and Systems, Tsinghua University, from 2009 to 2011. From 2011 to 2015, he was an Assistant Professor with the School of Aeronautics and Astronautics, Zhejiang University. He also was a Visiting Scholar with the Robotics Institute, Carnegie Mellon University, from 2013 to 2014. Since 2016, he has been an Associate Professor with the College of Control Science and Engineering, Zhejiang University. He is the author of more than 30 articles and more than ten inventions. His research interests include navigation guidance and control of aerial robots, visual navigation and environmental mapping, intelligent control theory and application, and artificial intelligence. He was a member of the Intelligent Automation Technical Committee of the China Association of Automation and the Zhejiang Association of Automation.

Xi Chen was born in Jinan, China, in 1973. He received the B.S. degree in chemical engineering, the M.S. degree in system engineering, and the Ph.D. degree in control science and engineering from Zhejiang University, China, in 1993, 1996, and 2001, respectively. Since 2001, he has been a Faculty Member of the Department of Control Science and Engineering, Zhejiang University, starting from an Assistant Professor (2001–2004), to an Associate Professor (2004–2010), and has been a Full Professor (since 2010). His research interests include process systems engineering, parallel computation, symbolic computation, and related industrial applications. He was a recipient of the Science and Technology Progress Award of Ministry of Education in China in 2005 and the Natural Science Award of Ministry of Education in China in 2013. He is an Associate Editor of the *Journal of Process Control*.