

Combining deep reinforcement learning with heuristics to solve the traveling salesman problem

Li Hong(洪莉), Yu Liu(刘宇), Mengqiao Xu(徐梦俏), and Wenhui Deng(邓文慧)

Citation: Chin. Phys. B, 2025, 34 (**1**): 018705. DOI: 10.1088/1674-1056/ad95f1

What follows is a list of articles you may be interested in

Optimized quantum singular value thresholding algorithm based on a hybrid quantum computer

Yangyang Ge(葛阳阳), Zhimin Wang(王治旻), Wen Zheng(郑文), Yu Zhang(张钰), Xiangmin Yu(喻祥敏), Renjie Kang(康人杰), Wei Xin(辛蔚), Dong Lan(兰栋), Jie Zhao(赵杰), Xinsheng Tan(谭新生), Shaoxiong Li(李邵雄), and Yang Yu(于扬)

Chin. Phys. B, 2022, 31 (**4**): 048704. DOI: 10.1088/1674-1056/ac40fb

Numerical simulations of partial elements excitation for hemispherical high-intensity focused ultrasound phased transducer

Yanqiu Zhang(张艳秋), Hao Zhang(张浩), Tianyu Sun(孙天宇), Ting Pan(潘婷), Peiguo Wang(王佩国), and Xiqi Jian(菅喜岐)

Chin. Phys. B, 2021, 30 (**7**): 078704. DOI: 10.1088/1674-1056/ac05a2

Modeling for heterogeneous multi-stage information propagation networks and maximizing information

Ren-Jie Mei(梅人杰), Li Ding(丁李), Xu-Ming An(安栩明), Ping Hu(胡萍)

Chin. Phys. B, 2019, 28 (**2**): 028701. DOI: 10.1088/1674-1056/28/2/028701

Treatable focal region modulated by double excitation signal superimposition to realize platform temperature distribution during transcranial brain tumor therapy with high-intensity focused ultrasound

Shi-Hui Chang(常诗卉), Rui Cao(曹睿), Ya-Bin Zhang(张亚斌), Pei-Guo Wang(王佩国), Shi-Jing Wu(吴世敬), Yu-Han Qian(钱宇晗), Xi-Qi Jian(菅喜岐)

Chin. Phys. B, 2018, 27 (**7**): 078701. DOI: 10.1088/1674-1056/27/7/078701

Image reconstruction from few views by l_0-norm optimization

Sun Yu-Li (孙玉立), Tao Jin-Xu (陶进绪)

Chin. Phys. B, 2014, 23 (**7**): 078703. DOI: 10.1088/1674-1056/23/7/078703

SPECIAL TOPIC — Computational programs in complex systems

Combining deep reinforcement learning with heuristics to solve the traveling salesman problem

Li Hong(洪莉)[1], Yu Liu(刘宇)[1], Mengqiao Xu(徐梦俏)[2,†], and Wenhui Deng(邓文慧)[2]

[1] *School of Software Technology, Dalian University of Technology, Dalian* 116620, *China*
[2] *School of Economics and Management, Dalian University of Technology, Dalian* 116024, *China*

(Received 27 July 2024; revised manuscript received 13 November 2024; accepted manuscript online 22 November 2024)

Recent studies employing deep learning to solve the traveling salesman problem (TSP) have mainly focused on learning construction heuristics. Such methods can improve TSP solutions, but still depend on additional programs. However, methods that focus on learning improvement heuristics to iteratively refine solutions remain insufficient. Traditional improvement heuristics are guided by a manually designed search strategy and may only achieve limited improvements. This paper proposes a novel framework for learning improvement heuristics, which automatically discovers better improvement policies for heuristics to iteratively solve the TSP. Our framework first designs a new architecture based on a transformer model to make the policy network parameterized, which introduces an action-dropout layer to prevent action selection from overfitting. It then proposes a deep reinforcement learning approach integrating a simulated annealing mechanism (named RL-SA) to learn the pairwise selected policy, aiming to improve the 2-opt algorithm's performance. The RL-SA leverages the whale optimization algorithm to generate initial solutions for better sampling efficiency and uses the Gaussian perturbation strategy to tackle the sparse reward problem of reinforcement learning. The experiment results show that the proposed approach is significantly superior to the state-of-the-art learning-based methods, and further reduces the gap between learning-based methods and highly optimized solvers in the benchmark datasets. Moreover, our pre-trained model M can be applied to guide the SA algorithm (named M-SA (ours)), which performs better than existing deep models in small-, medium-, and large-scale TSPLIB datasets. Additionally, the M-SA (ours) achieves excellent generalization performance in a real-world dataset on global liner shipping routes, with the optimization percentages in distance reduction ranging from 3.52% to 17.99%.

Keywords: traveling salesman problem, deep reinforcement learning, simulated annealing algorithm, transformer model, whale optimization algorithm

PACS: 87.55.kd, 87.55.de, 07.05.Mh

DOI: 10.1088/1674-1056/ad95f1 **CSTR:** 32038.14.CPB.ad95f1

1. Introduction

The traveling salesman problem (TSP) is a classic combinatorial optimization problem, which has received widespread attention in all walks of life. The TSP is represented as a problem where a salesman needs to visit all the cities in his area, with each city only being visited once, and then return to the starting city. The TSP has numerous real-word applications, such as transportation,[1] multi-robots,[2] logistics distribution,[3] and aviation planning.[4] Although researchers have conducted a great number of studies on the TSP, obtaining the shortest route to accomplish the travel remains challenging because of the NP-hard nature of this problem.[5]

Numerous approaches have been proposed to solve the TSP. Classical methods mainly involve the use of exact algorithms and heuristic algorithms.[6] Exact algorithms have dynamic programming methods, branch and bound methods, etc.[7] However, as the scale of the problem increases, exact algorithms may become helpless. Scholars thus focus on using heuristic algorithms, which mainly include greedy algorithms,[8] genetic algorithms,[9] simulated annealing algorithms,[10] particle swam optimization,[11] and whale optimization algorithms.[12] Although the heuristic algorithm lacks the theoretical guarantee, it can find the ideal solution within a reasonable calculation time.[13] Therefore, the heuristic algorithm is widely used in practice. In addition, heuristic algorithms require a large number of experimental processes, and the quality of the solution is highly dependent on domain knowledge.[14]

With the rapid development of artificial intelligence, reinforcement learning (RL), deep reinforcement learning (DRL) methods[15] and deep models (e.g., convolutional neural networks[16,17] and attention networks[18]) have been introduced to solve the TSP. The study in Ref. [19] uses three different RL algorithms, namely, Q-learning, Sarsa, and double Q-learning to solve the TSP. The study in Ref. [20] combines DRL and deep convolutional neural networks to provide real-time solutions for the TSP with node sizes up to 100. The study in Ref. [21] proposes a self-learning approach,

†Corresponding author. E-mail: stephanie1996@sina.com

© 2025 Chinese Physical Society and IOP Publishing Ltd. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

which combines DRL with a Monte Carlo tree search to solve the TSP. The study in Ref. [22] proposes a novel GCE-MAD model based on a graph convolutional encoder and attention decoder network for the TSP, and designs a DRL method to train the model. These methods have been widely recognized as able to solve the TSP with excellent performance. However, the use of DRL methods and deep models makes it difficult to solve the TSP with large-scale datasets, indicating that there is still a gap between the current DRL methods and the competitive heuristic algorithms.

Recently, some studies have combined DRL with heuristics to solve the TSP. The study in Ref. [23] presents a DRL framework with a learning 2-opt algorithm via a policy gradient for the TSP. The study in Ref. [24] develops a model based on an attention model and uses RL with a greedy strategy to train the model, achieving optimal results for node sizes up to 100 cities for the TSP. The study in Ref. [25] proposes a DRL to learn a local search method, which is trained by a policy gradient and introduces a pointer attention mechanism to sequentially output nodes for action selection. The study in Ref. [26] designs a novel approach called MAGIC, which is a DRL training method with local search heuristics (i.e., 2-opt and local insertion heuristics) to solve the TSP. The study in Ref. [27] combines machine learning techniques with a 2-opt heuristic algorithm to solve the TSP. This work presents an improvement to the computational weight of the original deep learning model to reduce the execution time and further improve performance. The study in Ref. [28] presents an average DRL approach to learn an improvement 2-opt heuristic algorithm to solve the TSP. The study in Ref. [29] proposes a novel DRL framework and designs a new policy network based on an attention mechanism to learn improvement of the 2-opt heuristic algorithm, which shows superiority in solving routing problems. These approaches are capable of incorporating heuristic algorithms and can obtain high-quality solutions. However, most methods focus on learning construction heuristics, which can generate complete solutions sequentially by adding nodes to partial solutions in each step. Such methods of learning construction heuristics still depend on additional programs (e.g., a beam search strategy or a sampling strategy) to improve the solution, which have limited capabilities because these programs depend on identical trained construction policies. Very few studies have focused on learning improvement heuristics that can iteratively refine a solution by learning policies. The traditional improved heuristic algorithms are guided by the manually designed search strategy, require a lot of expert knowledge, and may only achieve limited improvements. By comparison, the DRL can automatically learn better improvement policies for heuristic algorithms to iteratively solve the TSP.

The RL algorithm trains an agent to solve the TSP using data obtained from interactions between the agent and the environment. The optimal strategy of RL can only be learned from the data obtained from the interaction with the environment, since the dynamic model of the default environment is unknown.[30] Therefore, the balance between the exploration phase and the exploitation phase is the core of the RL algorithm.[31] Exploitation makes decisions based on the agent's existing knowledge of the environment, while exploration requires intentionally performing random or even suboptimal actions to escape the local optima. Obviously, the agent can only find the global optimal policy after fully exploring the environment, but abundant exploration will make the whole training process of the agent slow and unstable. The simulated annealing algorithm (SA)[32] is inspired by the principle of solid annealing in the process of metal heat treatment and was successfully applied to combinatorial optimization problems in 1983. The SA algorithm accepts a poor solution with a certain probability, and thus can escape the local optimum to approach the global optimum.[32] Hence, the SA algorithm can effectively help balance the exploration and exploitation phases of the RL algorithm.

This paper proposes a novel framework for learning improvement heuristic algorithms, which automatically discovers better improvement policies for heuristic algorithms to iteratively solve the TSP. Specifically, our framework firstly designs a new architecture based on a transformer model to parameterize the policy network that learns with pairwise operators for the 2-opt heuristic algorithm. This architecture, for the first time, introduces an action-dropout layer to prevent action selection from overfitting, which can increase the diversity of action selection. Then, our framework proposes a novel approach (named RL-SA) which, for the first time, combines deep RL with the SA algorithm to improve the 2-opt heuristic algorithm's performance in learning a pairwise selected policy of next promising solutions. The novelty of the RL-SA approach also lies in that it leverages the whale optimization algorithm (WOA)[33] to generate an initial solution for better sampling efficiency, and that it adopts the Gaussian perturbation strategy[34] to perturb the state to tackle the sparse reward problem of the RL algorithm. The experiment results show that the proposed approach is significantly superior to state-of-the-art learning-based methods, and further reduces the gap between learning-based methods and highly optimized solvers in benchmark datasets. Moreover, our pre-trained model M can be successfully applied to guide the SA algorithm (named M-SA (ours)), which has better optimization performance than existing deep model methods in the TSPLIB standard library with different-sized synthetic datasets. Note that the distribution of test data is completely different from that of model training data. Additionally, the M-SA (ours) method performs well in a real-world dataset on global liner shipping routes.

For each original shipping route, our method can provide an optimization route that effectively reduces the total shipping distance of the original route, which may help shipping companies achieve the goal of reducing fuel costs and thus increasing profits.

The structure of this paper is outlined as follows. Section 2 covers the preliminaries. Section 3 presents the proposed method. Section 4 details the primary experiments and results. Section 5 concludes with an analysis of the strengths and limitations of the proposed method, along with a discussion of future research directions.

2. Preliminaries

2.1. Traveling salesman problem

The TSP is a combinatorial optimization problem of NP-hard nature, which requires exponential time to obtain the optimal solution.[5] The TSP can be defined by the fact that it gives an input graph and is represented as a sequence of m nodes (cities) in a two-dimensional space $C = \{c_i\}_{i=1}^{m}$, where c_i is the coordinate and node features of the i-th location. We are concerned with finding a sequence of the nodes, which is called a tour solution $X = (x_1, \ldots, x_m)$, such that each node, except for the start node, should be visited only once. This paper considers the minimal Euclidean distance of the route according to previous research. The minimal Euclidean distance, denoted as $f(X)$, is expressed as

$$f(X) = \sum_{i=1}^{m-1} ||C_{x_i} - C_{x_{i+1}}||_2 + ||C_{x_m} - C_{x_1}||_2, \tag{1}$$

where $||\cdot||_2$ denotes the $L2$-norm.

2.2. Basic theory of the SA

The theory of the SA algorithm[32] is described as heating the solid to a certain temperature and allowing the temperature to cool slowly. During this process, the particles become disordered and the internal energy increases when the temperature rises. The particles gradually become ordered when the temperature cools slowly. The equilibrium state is reached at each temperature, and the energy is minimized when it is finally completely cooled.

According to the Metropolis criterion,[35] the probability P_i of the particles tending to equilibrium at the current temperature T_i can be expressed as

$$P_i = \exp(-\Delta E / T_i), \tag{2}$$

$$T_{i+1} = k \times T_i, \tag{3}$$

where k denotes the temperature decay rate that is a number in (0, 1), E denotes the energy at temperature T_i and ΔE denotes the energy change.

A classic SA algorithm for an optimization problem is given by the following steps.

Step 1 Generate a random initial solution, find the optimal solution, and initialize the temperature.

Step 2 Get a new solution by searching near the current solution.

Step 3 Evaluate the new solution, by calculating the objective function of the new solution (denoted as E_{i+1}) and comparing it to the objective function of the current solution (denoted as E_i). If the new solution is a better value, or if the probability function (Eq. (2)) is a higher value than a randomly generated number in (0, 1), the current solution will be updated. Otherwise, keep the current solution selected.

Step 4 Update the temperature via Eq. (3).

Step 5 Repeat steps 2–5 until termination conditions are met, and the optimal solution is output.

2.3. Basic theory of the WOA

The WOA, proposed by Mirjalili *et al.* in 2016, is a novel swarm-intelligence metaheuristic algorithm inspired by the bubble-net predation behavior of humpback whales.[33] Humpback whales' unique bubble-net predation behavior includes encircling prey, the spiraling attacking of prey and the searching for prey.

(i) Encircling prey: the whales are able to observe their preys' positions (e.g., krill, fish) and achieve complete encircling of them. The encircling prey behavior is represented as

$$s = s^{\text{best}} - W \cdot d, \tag{4}$$

$$d = \left| 2 \cdot r \cdot s^{\text{best}} - s \right|, \tag{5}$$

$$W = w \cdot (2 \cdot r - 1), \tag{6}$$

where d is the vector representing the distance between the whales and the current best whale; w denotes the adjustment coefficient, and its value from 2 to 0 linearly decreases with iterations increasing; W ($\in [-w, w]$) represents the parameter vector; s represents the whales' position vector; s^{best} represents the current best whale's position vector; r ($\in [0, 1]$) represents a random number; and $|\ |$ denotes the absolute value.

(ii) Bubble-net attacking method: the encircling prey behavior and the spiraling attacking prey behavior are adopted to simulate the bubble-net attacking method of humpback whales. Achieving the encircling prey behavior is via changing w according to Eq. (6). If $|W| \leq 1$ then, according to Eq. (4), the whales will choose the best whale for a position update. The spiraling attacking prey behavior describes the whales attacking their prey in a logarithmic spiral and is represented as

$$s = \cos(2 \cdot \pi \cdot l) \cdot \mathrm{e}^{b \cdot l} \cdot d' + s^{\text{best}}, \tag{7}$$

$$d' = \left| s^{\text{best}} - s \right|, \tag{8}$$

where l ($\in[-1,1]$) represents a random number, d' is the vector representing the distance between the whales and the current best whale, b denotes a constant value, which defines the logarithmic spiral shape, and b represents 1 in the present study.

Note that the humpback whales simultaneously use two behavioral patterns of the encircling and the spiraling model to update their positions. To achieve this synchronous behavior, the WOA performs these two position update behavioral patterns via the parameter p and is represented as

$$s=\begin{cases} s^{\text{best}}-W\cdot d, & \text{if } p\le\beta,\\ \cos(2\cdot\pi\cdot l)\cdot \mathrm{e}^{b\cdot l}\cdot d'+s^{\text{best}}, & \text{if } p>\beta, \end{cases} \tag{9}$$

where p ($\in[0,1]$) denotes a random number, and β represents a constant whose value is 0.5 in the present study.

(iii) Searching for prey: the searching for prey behavior is controlled by changes in the coefficient vector W in the process of hunting. If $|W|\le 1$, the whales will choose the best whale for a position update. If $|W|>1$, the whales will abandon the best whale and choose a whale at random to replace the current best whale for a position update. The searching for prey behavior is represented as

$$s=s_{\text{r}}-W\cdot d, \tag{10}$$

$$d=|2\cdot r\cdot s_{\text{r}}-s|, \tag{11}$$

where s_{r} represents a randomly chosen whale's position vector.

3. The method

This section first formulates the improvement heuristics for iteratively solving the TSP as an RL task. Then, an architecture based on a transformer model is designed to make the policy network parameterized. Finally, an RL-SA method is proposed to improve the 2-opt heuristic algorithm that can learn a pairwise selected policy of the next solution.

3.1. RL formulation

The task of improving heuristics to solve the TSP is formulated as a Markov decision process (MDP).[36] First, a complete initial solution is generated. Then, the neural network policy iteratively improves the solution's quality. This policy automatically selects a node pair and adjusts the solution via the node pair operator (i.e., 2-opt) at each step.

The 2-opt describes the random selection of two non-adjacent nodes from the solution and swaps them to obtain a new solution. As an example, suppose that there are m cities, which are labeled $x_1,x_2,x_3,x_4,x_5,x_6,\ldots,x_m$ respectively. Suppose that the current optimal solution is s_t, i.e., $s_t=\{x_1^t,x_2^t,x_3^t,x_4^t,x_5^t,x_6^t,\ldots,x_m^t\}$. The 2-opt policy randomly selects two non-adjacent nodes x_2^t and x_5^t in s_t. Then, the node sequence before node x_2^t and after node x_5^t remains unchanged, and the node sequence between node x_2^t and node x_5^t is inverted to add a new solution, i.e., $s_{t+1}=\{x_1^{t+1},x_5^{t+1},x_4^{t+1},x_3^{t+1},x_2^{t+1},x_6^{t+1},\ldots,x_m^{t+1}\}$.

Specifically, the MDP consists of the following components.

State: the state s_t indicates the solution at time t for an instance with m nodes, i.e., $s_t=\{x_1^t,x_2^t,\ldots,x_m^t\}$.

Action: the action a_t indicates a node pair in the current state s_t for the pairwise operator, i.e., $a_t=\{x_i^t,x_j^t\}$.

Reward: the reward function r_t is designed as follows: $r_t=f(s_t^*)-\min[f(s_{t+1}),f(s_t^*)]$, where s_t^* is the current optimal solution until time t. If s_{t+1} is better, we will update the current optimal solution, i.e., $f(s_t^*)>f(s_{t+1})$. The reward $r_t>0$ only if we find a better solution, otherwise the reward r_t is 0. Therefore, the maximized cumulative reward is represented as $R=\sum_{t=0}^{T}\gamma^t r_t$, where T denotes the maximum time, and γ denotes the discount factor that is a number in $[0,1]$.

Policy: the policy π is to select an action a_t in state s_t and leads to s_{t+1}, starting from the initial solution s_0 and until it reaches the maximum time. Specifically, this process can be represented by the rule of probability chaining, i.e., $P(s_T\,|\,s_0)=\prod_{t=0}^{T}\pi(a_t|s_t)$.

Transition: the state s_t transfers the next state s_{t+1} according to the preset pairwise on the give action a_t. The state transition is implemented according to the Metropolis criterion of the SA algorithm.

3.2. Policy network

A novel model based on a transformer,[18] named M, is proposed to make the policy network parameterized. The neural network π_θ is used to parameterize the model M and the θ is the trainable parameter for learning the policy π. The architecture of M consists of two parts, which are an encoder and a decoder, as shown in Fig. 1. The encoder embeds node features and positional features of each node. The decoder generates a probability matrix for selecting each node pair. The details of the encoder and decoder are as follows.

(i) Encoder: the encoder takes the node and position information as the input, and outputs the features (or embedding) of each node. Specifically, it first converts the information of each node into embeddings e_i through a linear projection (transformation) with the output dimension $d_n=128$. The position information of each node is converted into the embedding p_i by sinusoidal positional encodings,[18] given that the position is a sequence. The sinusoidal positional encoding is expressed by a sine function and a cosine function, which is shown as follows:

$$p(\text{pos},d)=\sin\left(\frac{\text{pos}}{10000^{d/d_n}}\right),\ \text{if}\ (d \bmod 2)=0, \tag{12}$$

$$p(\text{pos}, d) = \cos\left(\frac{\text{pos}}{10000^{d/d_n}}\right), \text{ if } (d \bmod 2) = 1, \quad (13)$$

where pos denotes the position of the node, d denotes the dimension, and mod denotes the modulo function. We fuse the node and positional information into a unified representation, i.e., $f_i = e_i + p_i$.

Next, to promote the node embeddings, the self-attention and the feed-forward layers are adopted, and the residual connection layer and batch normalization layer follow them. The above process is successively executed N $(N = 3)$ times with the same structure but different parameters, to obtain better feature representations for policy learning. Finally, the self-attention layer advances node embeddings by messaging and aggregation between them. By considering an input matrix F^{in} in which the column is node embeddings, the output of the self-attention layer is calculated using

$$F^{\text{out}} = \text{softmax}\left(\frac{Q \cdot K^{\text{T}}}{\sqrt{d_n}}\right) \cdot V, \quad (14)$$

$$Q = W^q \cdot F^{\text{in}}, \quad (15)$$

$$K = W^k \cdot F^{\text{in}}, \quad (16)$$

$$V = W^v \cdot F^{\text{in}}, \quad (17)$$

where Q, K and V respectively indicate the query, key, and value of the input matrix F^{in}. The W^q, W^k, and $W^v \in R^{d_n \times d_n}$ are trainable parameters. The feed-forward layer uses parameter sharing to independently transform each embedded node. The input and the output are 128 dimensions. We adopt the 128-dimensional hidden layer with the activation function of the ReLU.

(ii) Decoder: the decoder takes the node embeddings as input and further processes the input via a self-attention compatibility layer, an action-dropout layer, and a masked-softmax layer. The decoder outputs a probability matrix for each node pair selected. The self-attention compatibility layer, which is inspired by Ref. [18], is used to calculate the probability value of each node pair being chosen in the solution. By considering a node embedding matrix F, the self-attention compatibility function S, which indicates the value of each node pair selected, is calculated as $S = Q_F K_F^{\text{T}}$, where Q_F and K_F are computed similar to Q and K in the above content. Then, the action-dropout layer, inspired by Ref. [37], uses dropout as a technique to improve neural networks by reducing overfitting. The action-dropout layer is adopted to prevent action selection from overfitting, which can increase the diversity of action selection. The action-dropout layer S^{∂} is calculated using

$$S^{\partial} = r \cdot S, \quad (18)$$

$$r \sim \text{Bernoulli}(p), \quad (19)$$

where r denotes a vector of independent Bernoulli random variables, each of which has probability p $(p = 0.5)$. Finally, the masked-softmax layer masks the meaningless diagonal elements and uses softmax for the compatibility matrix S_{ij}^{∂}, where selecting a node pair (i, j) is an action. First, $S_{ij}^{\sim\partial} = u \cdot \tanh(S_{ij}^{\partial})$ is applied to restrain the entropy of $S_{ij}^{\sim\partial}$, where the value u is 10. Then, the softmax function is adopted to normalize the action distribution and obtain the probability value of selecting the node pair as $P_{ij} = \text{softmax}(S_{ij}^{\sim\partial})$.

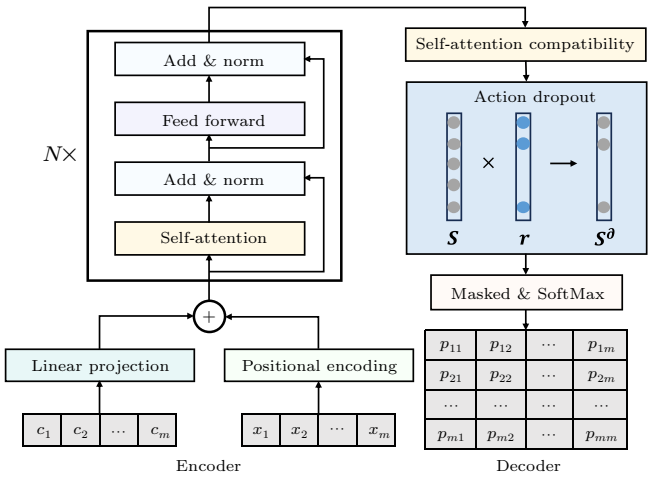


Fig. 1. The M model for the policy network.

3.3. RL-SA method

A novel RL-SA method is proposed, which combines the actor-critic algorithm of RL with the SA algorithm to train the model M (i.e., learn the pairwise selected policy), aiming to improve the 2-opt heuristic algorithm. Then, the training method leverages the WOA algorithm to generate initial solutions for better sampling efficiency. Furthermore, the Gaussian perturbation strategy is used to perturb the state, increase the diversity of solutions and deal with the sparse reward problem of the RL algorithm.

The actor-critic algorithm with an Adam optimizer belongs to the policy gradient method based on the REINFORCE algorithm.[38] The actor is our policy network M. The critic π_τ, which is similar to the policy network M, is used to estimate the cumulative reward in each state.

The RL algorithm could find the global optimal strategy only after substantially exploring the environment. However, a large number of explorations will make the whole training process slow and unstable. The SA algorithm accepts a poor solution with a certain probability, and thus can escape from the local optimum. Hence, the SA algorithm is integrated into our method to help effectively balance the exploration and exploitation phases of the RL algorithm.

Next, when considering the cost of training, the training steps are usually much smaller than actual reasoning steps, which may result in a poor chance of the agent observing high-quality solutions during the training. Hence, the WOA algorithm is used to initialize solutions that specify a higher-quality

solution as the initial state of training, for better sampling efficiency. For a comparison of the quality of initial solutions generated by the WOA and random methods, see Appendix A.

Finally, due to the large state space, the agent may encounter a sparse reward problem in exploring the environment when using the RL algorithm to solve real-world problems like the TSP.[39] The sparse reward problem refers to how difficult it is for agents to obtain positive rewards in the exploration phase, which results in slow learning or even inability to learn. Hence, the Gaussian perturbation strategy is used to perturb the state to increase the diversity of solutions and solve the sparse reward problem. The Gaussian perturbation strategy is represented as

$$s = s^{\text{best}} + \varepsilon \cdot \delta, \tag{20}$$

where δ represents a random vector following a standard normal distribution, i.e., $\delta \sim N(0,1)$, which is the same dimension as s^{best}, and ε represents the weight of Gaussian perturbation and is a constant value in $(0,1)$.

The pseudocode for the RL-SA algorithm is represented as follows in Algorithm 1.

Algorithm 1: The pseudo-code for the RL-SA algorithm.

1. begin
2. Input: initial and termination temperature T_{i} and T_{e}, temperature decay rate, initialize the parameters of the WOA, actor and critic network π_θ and π_τ with trainable parameters θ and τ, epochs E, batches B, discount factor γ;
3. for each epoch
4. Generate problem instances randomly;
5. for each batch
6. Retrieve each batch b of instances and adopt the WOA to generate the initial states (solutions);
7. while ($T_{\text{i}} > T_{\text{e}}$)
8. Set the gradient $\mathrm{d}\theta \leftarrow 0$ and $\mathrm{d}\tau \leftarrow 0$;
9. Get the state s_t at time step t;
10. Sample the action a_t by the policy $\pi_\theta(a_t|s_t)$;
11. According to the Metropolis criterion of the SA for the next state s_{t+1};
12. Adopt a Gaussian perturbation strategy to perturb s_{t+1}, and receive reward r_t;
13. Lower the temperature via Eq. (3);
14. $\varphi \leftarrow r_t + \gamma\pi_\tau(s_t) - \pi_\tau(s_{t-1})$;
15. $\mathrm{d}\theta \leftarrow \mathrm{d}\theta + \sum_b \varphi \nabla \log \pi_\theta(a_t|s_t)$
16. $\mathrm{d}\tau \leftarrow \mathrm{d}\tau + \sum_b \varphi \nabla \pi_\tau(s_t)$;
17. Update θ and τ via $\mathrm{d}\theta$ and $\mathrm{d}\tau$;
18. end
19. end
20. end
22. end

4. Experiments and results

Our method is evaluated on a typical and representative routing problem, which is the TSP. The instances are randomly generated with 20, 50, and 100 nodes abiding by existing conventions. The instances are generated according to Refs. [24,29,40], where the nodes' coordinates are randomly sampled in the unit square, i.e., $[0,1] \times [0,1]$, obeying a uniform distribution. A total of 10000 random instances of the TSP are generated dynamically in each training epoch. We adopt the WOA to generate the initial solutions for training and the solution generated by random initial solutions for reasoning. Furthermore, our model can be successfully applied to guide the SA algorithm, which outperforms existing deep-learning-based methods in the TSPLIB standard library[41] with different-sized synthetic datasets and exhibits a better performance on a real-world dataset on global liner shipping routes.[42]

In our method, the parameters are set as follows: the discount factor γ is 0.95; the batch size B is 512; the epochs E is 200; the initial learning rate is 10^{-4} and the attenuation factor is 0.99 every epoch to achieve convergence; the initial and termination temperature are 100 and 0.01, respectively; the temperature decay rate is 0.9; and the weight of Gaussian disturbance ε is 0.35. Our model M is trained and tested on a server equipped with GeForce RTX GPU cards and Intel i7-12700 CPU at 2.10 GHz.

4.1. Comparison with state-of-the-art methods

Our method is compared with various advanced baselines: (i) LKH3,[43] which is a famous heuristic solver for solving various routing problems and achieving advanced performance; (ii) Concorde,[44] which is an efficient accurate solver; (iii) OR-Tools,[45] which is a mature solver based on meta-heuristics; (iv) AM (sampling),[24] which is based on a deep learning construction method; (v) Lei *et al.* (sampling),[46] which is a competitive method and is based on a deep reinforcement learning method; (vi) Wu *et al.*,[29] which is based on a learning improvement heuristic method.

Table 1 shows the optimal solutions and the total time for reasoning the 10000 instances of these methods. The gap (%) is calculated based on the optimal solutions by the LKH3 (or Concorde) method. The results in bold indicate the best performance achieved by learning-based methods.

As Table 1 shows, for all the problems, the gap between our method and the optimal solution gradually narrows as the reasoning step T limit increases ($T = 1000$, 3000 and 5000). When $T = 1000$, our method is significantly better than the OR-Tools and Wu *et al.* When $T = 3000$, our method outperforms AM and Lei *et al.* When $T = 5000$, our method is superior to learning-based methods for all the problems; in particular, it achieves the same results as the LKH3 and Concorde methods in solving the optimal solution for TSP20 and TSP50 problems. Note that the gap between our method and the LKH3 (or Concorde) method in solving the optimal solution for the TSP100 problem is 1.16%, and our method per-

forms better than other methods in this problem. Furthermore, our method also achieves good performance in reasoning time, which is faster than Lei *et al.* and Wu *et al.* for a reasoning time of 10000 instances for all problems. Its reasoning time grows much slower than other methods with the increase in problem size. Compared to LKH3, Concorde, AM (sampling) and OR-Tools, ours ($T = 1000$) are slower for TSP20 problems but are faster (or at least not slower) for TSP100 problems.

Table 1. Comparison with the state-of-the-art methods.

Methods	TSP20			TSP50			TSP100		
	Obj.	Gap (%)	Time	Obj.	Gap (%)	Time	Obj.	Gap (%)	Time
LKH3	3.83	0	42 s	5.69	0	6 min	7.76	0	25 min
Concorde	3.83	0	5 min	5.69	0	13 min	7.76	0	1 h
OR-Tools	3.86	0.78	1 min	5.85	2.81	5 min	8.06	3.87	23 min
AM (sampling)	3.84	0.26	5 min	5.73	0.70	24 min	7.94	2.32	1 h
Lei *et al.* (sampling)	**3.83**	**0**	10 min	5.738	0.84	55 min	7.896	1.75	2 h
Wu *et al.* ($T = 1000$)	**3.83**	**0**	12 min	5.74	0.88	16 min	8.01	3.22	25 min
Wu *et al.* ($T = 3000$)	**3.83**	**0**	39 min	5.71	0.35	45 min	7.91	1.93	1.5 h
Wu *et al.* ($T = 5000$)	**3.83**	**0**	1 h	5.70	0.18	1.5 h	7.87	1.42	2 h
Ours ($T = 1000$)	**3.83**	**0**	8 min	5.71	0.35	13 min	7.94	2.32	23 min
Ours ($T = 3000$)	**3.83**	**0**	23 min	5.70	0.18	38 min	7.87	1.42	1 h
Ours ($T = 5000$)	**3.83**	**0**	39 min	**5.69**	**0**	1 h	**7.85**	**1.16**	1.8 h

Note: the results in bold indicate the best performance achieved by learning-based methods.

4.2. Application analysis of the M model

The traditional improved heuristic algorithms are guided by a manually designed search strategy, require a lot of expert knowledge, and may only obtain limited improvements.[14,47] Recently, researchers have been focusing on utilizing deep learning methods to automatically learn better improvement policies without requiring domain knowledge for heuristic algorithms.[24,48,49] Meanwhile, when considering the high cost of model training, the application of a pre-trained model is crucial. The SA is a classical method used to solve combinatorial optimization problems. Compared with the 2-opt algorithm, the SA is an algorithm with more sophisticated rules. Hence, to further improve the performance of our proposed method, our pre-trained model M is used to guide the SA algorithm, which we term the M-SA (ours) method.

Indeed, the M-SA (ours) outperforms existing deep-learning-based methods in the TSPLIB standard library[41] with different-sized synthetic datasets. The distribution of the test data is completely different from that of the model training data. Table 2 shows the results from solving 36 instances in the small- and medium-scale TSPLIB datasets. The Opt. represents the known optimum. The OR-Tools is a strong conventional solver. The SA is a classic heuristic algorithm and indicates the minimum results of 10 runs. The AM is a learning-based construction method. The Wu *et al.* method is a learning-based improvement method. The M-SA (ours) indicates the minimum results of 10 runs. The Avg. gap (%) is calculated based on the known optimal solutions for 36 instances. The M-SA (ours) adopts the policy trained on TSP50. The AM and Wu *et al.* methods use the policy trained on TSP100. The best results among these three learning-based methods are denoted in bold.

As Table 2 shows, for the 36 instances, the average gap of OR-Tools, SA, AM, Wu *et al.*, and M-SA (ours) compared to the known optimum is 3.46%, 9.86%, 133.53%, 17.15%, and 7.34%, respectively. The M-SA (ours) is superior to the SA, which indicates our model can provide a better strategy for the SA. All the learning-based methods are inferior to the OR-Tools method, since obtaining good out-of-distribution generalization is usually very difficult for machine learning models. But it is worthwhile to note that the M-SA (ours) is superior to OR-Tools for Eil51, Berlin52, Pr76, KroC100, Pr124, and KroB150 instances, and that the M-SA (ours) significantly outperforms the AM and Wu *et al.* methods in the 36 instances.

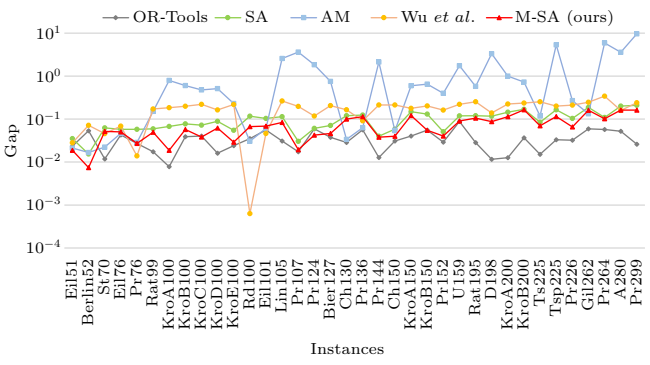


Fig. 2. The gap between the five methods and the known optimum for 36 instances.

To better display the comparison results of several methods in Table 2, figure 2 plots, on a log scale, the gap values of the OR-Tools, SA, AM, Wu *et al.*, and M-SA (ours) methods for each of the 36 instances. One can clearly observe that for the 36 instances, the gap between our method and the known

optimum is generally small, compared with the gap between the SA and the other two learning-based algorithms and the known optimum. Meanwhile, the performance of our method is relatively stable with the increase in the problem node size.

Moreover, to test our method on large-scale datasets, we further selected, from TSPLIB datasets, 18 instances with node sizes up to 2392. As Table 3 shows, the average gap of Wu *et al.* and M-SA (ours) compared to the known optimum is 24.84% and 20.51%, respectively. Hence, our method is advantageous over some existing state-of-the-art methods like Wu *et al.* in large-scale datasets, although the advantage is smaller than that on small-scale and medium-scale datasets. For the details of the results obtained via the Wu *et al.* method, see Appendix B.

Table 2. Performance comparison of methods on small-scale and medium-scale TSPLIB datasets.

Instance	Opt.	OR-Tools	SA	AM	Wu *et al.*	M-SA (ours)
Eil51	426	436	441	435	438	**434**
Berlin52	7542	7945	7658	7668	8082	**7598**
St70	675	683	717	**690**	706	710
Eil76	538	561	569	**563**	575	565
Pr76	108159	111104	114419	111250	**109668**	111,085
Rat99	1211	1232	1283	1394	1419	**1271**
KroA100	21282	21448	22716	38200	25196	**21683**
KroB100	22141	23006	23862	35511	26563	**23412**
KroC100	20749	21583	22252	30642	25343	**21545**
KroD100	21294	21636	23192	32211	24771	**22615**
KroE100	22068	22598	23281	27164	26903	**22712**
Rd100	7910	8189	8832	8152	**7915**	8439
Eil101	629	664	694	667	**658**	672
Lin105	14379	14824	16021	51325	18194	**15579**
Pr107	44303	45072	45647	205519	53056	**45162**
Pr124	59030	62519	62631	167494	66010	**61541**
Bier127	118282	122733	126662	207600	142707	**123771**
Ch130	6110	6284	6842	**6316**	7120	6722
Pr136	96772	102213	108823	**102877**	105618	107784
Pr144	58537	59286	60884	183583	71006	**60763**
Ch150	6528	6729	6910	6877	7916	**6788**
KroA150	26524	27592	30479	42335	31244	**29728**
KroB150	26130	27572	29546	43114	31407	**27564**
Pr152	73682	75834	77441	103110	85616	**76641**
U159	42080	45778	47081	115372	51327	**45890**
Rat195	2323	2389	2602	3661	2913	**2568**
D198	15780	15963	17623	68104	17962	**17152**
KroA200	29368	29741	33609	58643	35958	**32713**
KroB200	29437	30516	34418	50867	36412	**34281**
Ts225	126643	128564	137292	141628	158748	**135500**
Tsp225	3916	4046	4564	24816	4701	**4367**
Pr226	80369	82968	88682	101992	97348	**85632**
Gil262	2378	2519	2823	**2693**	2963	2759
Pr264	49135	51954	54488	338506	65946	**54135**
A280	2579	2713	3096	11810	**2989**	2997
Pr299	48191	49447	58290	513673	59786	**56027**
Avg. gap (%)	0	3.46	9.86	133.53	17.15	**7.34**

Note: the best results among these three learning-based methods are denoted in bold.

Table 3. Performance comparison between two learning-based methods on large-scale TSPLIB datasets.

Instance	Opt.	Wu *et al.*	M-SA (ours)
Lin318	42029	52473	**49316**
Rd400	15281	19059	**17951**
Pr439	107217	130178	**128427**
Pcb442	50778	61754	**59125**
Fl417	11861	14970	**13590**
D493	35002	43114	**41031**
U574	36905	46659	**45512**
Rat575	6773	8377	**8141**
P654	34643	43068	**42832**
D657	48912	61673	**60631**
U724	41910	54963	**50779**
Rat783	8806	11232	**11110**
Pr1002	259045	315435	**310462**
Rl1323	270199	332012	**311908**
Fl1400	20127	26935	**25653**
Fl1577	22249	27773	**27627**
D1655	62128	74893	**74160**
Pr2392	378032	461203	**460069**
Avg. gap (%)	0	24.84	**20.51**

Note: the best results among these two learning-based methods are denoted in bold.

4.3. Test on a real-world dataset on liner shipping routes

About 80% of global trade is transported by sea; thus, the maritime transport system is important to the world economy. Rising and volatile fuel prices are a serious concern for shipping companies.[50] Fuel cost is one main factor influencing the configuration of liner shipping routes, which accounts for more than 60% of the ship's operating costs. For any given route, the fuel cost mainly depends on the total shipping distance of the route.[51] Hence, our goal is to find the route configuration with the shortest total shipping distance, in order to recommend the optimal route configuration for the container ships and thus reduce the operating costs of shipping companies. The M-SA (ours) method is adopted to solve such a TSP in a real-world dataset on liner shipping routes. Our method can effectively find an optimal route configuration that significantly reduces the total shipping distance of a route, compared with the original route configuration.

Table 4 shows the original shipping distances of the studied routes (km), which have node sizes (i.e., number of ports) up to 31. All these six intercontinental routes are deployed with large container cargo capacity (in twenty-foot equivalent units, TEU), indicating their particular importance in global liner shipping practices. Data on global liner shipping routes are sourced from Ref. [42]. Note that the routing problem is a combinatorial optimization problem and generally belongs to NP-hard problems.[5] Although exact methods provide good theoretical guarantees to achieve the optimal solution, they are actually limited to small instances (i.e., the node size is less than 28) due to their exponential complexity in the worst

case.[52,53]

Specifically, the M-SA (ours) method outputs an optimal route that keeps the same departure port as the original shipping route, ensures that each port in the original route is visited, and finally returns to the departure port.

For each shipping route, we run the M-SA (ours) 10 times, and report the average and standard deviation of the route distance optimization percentage (i.e., the percentage by which the original shipping distance is reduced). As reported in Table 4, route 4 has the smallest average distance optimization percentage value (3.52%), and route 1 has the largest value (17.99%). The standard deviations of the six routes are between 0 and 0.38%. Therefore, our method performs well on real-world datasets on liner shipping routes.

Table 4. Optimization results of six intercontinental liner shipping routes over 10 runs.

Route	Cargo capacity deployment (TEU)	Number of ports	Original shipping distance (km)	Shipping distance optimization percentage (%)	
				Average	Standard deviation
1	179550	31	73779.41	17.99	0.38
2	66276	28	66678.94	12.88	0.33
3	61737	28	58324	12.24	0
4	77602	29	59056.68	3.52	0
5	71190	29	58855	3.65	0
6	68684	28	58258.91	3.61	0

5. Conclusions

This article presents a novel framework for a learning improvement heuristic algorithm to solve the TSP iteratively. Our framework designs a novel network model based on a transformer model to make the policy network parameterized, and introduces an action-dropout layer to prevent action selection from overfitting. Then, our framework proposes an approach (named RL-SA) to learn the pairwise selected policy, aiming to improve the 2-opt heuristic algorithm's performance. The RL-SA approach adopts the WOA to generate initial solutions for better sample efficiency and uses the Gaussian perturbation strategy to perturb the state to tackle the sparse reward problem of the RL. Our experiments verify that our approach is significantly superior to state-of-the-art learning-based methods, and further reduces the gap between learning-based methods and highly optimized solvers in benchmark datasets. Besides, our pre-trained model M can be ably applied to guide the SA algorithm (named M-SA (ours)), which has better optimization performance than existing deep model methods in small-, medium- and large-scale TSPLIB datasets. In the 36 small-scale and medium-scale instances, the average gap between the M-SA (ours) and the known optimum narrowed to 7.34%. In the 18 large-scale instances, the M-SA (ours) also proves to be slightly advantageous over existing cutting-edge methods. Additionally, the M-SA (ours) method performs well in a real-world dataset on global liner shipping routes. For each original shipping route, our method can provide an optimization route that effectively reduces the total shipping distance of the original route, which may help shipping companies achieve the goal of reducing fuel costs and thus increasing profits.

In further studies, we will consider designing the M-SA (ours) method in order to solve the capacitated vehicle routing problem [54] by realizing further extension of our method.

Appendix A: Comparison of the use of WOA and random methods to generate initial solutions of TSPs

The WOA and random methods are used to generate initial solutions for TSP20, TSP50 and TSP100 problems, with 10000 random instances for each problem. Figure A1 shows the average values of the initial solutions of 10000 instances via two methods for problems of different scale, which clearly indicates that the WOA outperforms the random method in the generation of initial solutions.

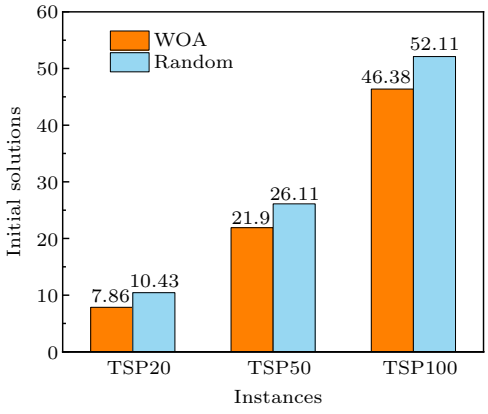


Fig. A1. Comparison between initial solutions of TSPs of different scales generated by WOA and random methods.

Appendix B: Details regarding the results for 18 large-scale instances obtained using the Wu *et al.* method.

Table 2 shows that the Wu *et al.* method[29] is a highly competitive method. Wu *et al.*[29] only report the results on small-scale and medium-scale TSPLIB datasets.

To obtain the results using the Wu *et al.* method on large-scale datasets, we retrain the Wu *et al.* model (the code is accessed from https://github.com/WXY1427/Learn-Improvement-Heuristics-for-Routing). We use the policy trained on TSP100, for which the reasoning step T is 3000; with that, we obtain the results on small-scale and medium-scale TSPLIB datasets, as shown in Table B1. For the 36 instances, Wu *et al.* report that the average gap compared to the known optimum is 17.15%, and the result reproduced by us compared to the known optimum is 15.82%. That is, the model we trained based on the Wu *et al.* method[29] successfully achieves (which is even better than) the performance reported in their paper. Therefore, based on this model we have trained, the results we obtained for the 18 large-scale TSPLIB instances under the Wu *et al.* method[29] (in Table 3) are credible.

Table B1. The results reported in Wu *et al.*[29] and the results reproduced by us.

Instance	Wu *et al.* (results in the paper)	Wu *et al.* (results reproduced)
Eil51	438	462
Berlin52	8082	7822
St70	706	703
Eil76	575	616
Pr76	109668	119623
Rat99	1419	1410
KroA100	25196	24614
KroB100	26563	25644
KroC100	25343	23209
KroD100	24771	24746
KroE100	26903	25463
Rd100	7915	9477
Eil101	658	738
Lin105	18194	18599
Pr107	53056	46176
Pr124	66010	66696
Bier127	142707	125461
Ch130	7120	7134
Pr136	105618	114850
Pr144	71006	61448
Ch150	7916	7862
KroA150	31244	31617
KroB150	31407	31532
Pr152	85616	82237
U159	51327	52018
Rat195	2913	2653
D198	17962	18430
KroA200	35958	34519
KroB200	36412	36012
Ts225	158748	144408
Tsp225	4701	4723
Pr226	97348	92006
Gil262	2963	3168
Pr264	65946	55713
A280	2989	3110
Pr299	59786	59645
Avg. gap (%)	17.15	15.82

Program availability

The source code is openly available in Science Data Bank at https://doi.org/10.57760/sciencedb.16742.

Acknowledgment

Project supported by the National Natural Science Foundation of China (Grant Nos. 72101046 and 61672128).

References

[1] Tirkolaee E B, Cakmak E and Karadayi-Usta S 2024 *International Transactions in Operational Research* **0** 13452
[2] Lamperti R D and de Arruda L V R 2023 *Engineering Applications of Artificial Intelligence* **126** 106884
[3] Zhang X X 2020 *Proceedings of the Sixth International Forum on Decision Sciences* (Singapore: Springer) p. 65
[4] Xu Y and Che C 2019 *IEEE 9th International Conference on Electronics Information and Emergency Communication*, July 12–14, 2019, Beijing, China, p. 7
[5] Si J, Yang S, Cen Y, *et al.* 2024 *Nat. Commun.* **15** 3457
[6] Cheikhrouhou O and Khoufi I 2021 *Computer Science Review* **40** 100369
[7] Yang L, Wang X, He Z, *et al.* 2023 *International Conference on Bio-Inspired Computing: Theories and Applications*, December, 2023, Singapore, p. 3
[8] Kizilateş G and Nuriyeva F 2013 *Advances in Computational Science, Engineering and Information Technology* (Heidelberg: Springer) p. 111
[9] Snyder L V and Daskin M S 2006 *European Journal of Operational Research* **174** 38
[10] Ezugwu A E S, Adewumi A O and Frîncu M E 2017 *Expert Systems with Applications* **77** 189
[11] Cui Y, Zhong J, Yang F, *et al.* 2020 *IEEE Access* **8** 227497
[12] Zhang J, Hong L and Liu Q 2020 *Symmetry* **13** 48
[13] Juan A A, Kelton W D, Currie C S M, *et al.* 2018 *Winter Simulation Conference*, December 9–12, 2018, Gothenburg, Sweden, p. 3048
[14] Khalil E, Dai H, Zhang Y, *et al.* 2017 *Advances in Neural Information Processing Systems*, December, 2017, Long Beach, California, USA, p. 6351
[15] Arulkumaran K, Deisenroth M P, Brundage M, *et al.* 2017 *IEEE Signal Processing Magazine* **34** 26
[16] Li Z, Liu F, Yang W, *et al.* 2021 *IEEE Transactions On Neural Networks And Learning Systems* **33** 6999
[17] Ling Z, Tao X, Zhang Y, *et al.* 2020 *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **51** 7475
[18] Vaswani A, Shazeer N, Parmar N, *et al.* 2017 *Advances in Neural Information Processing Systems*, 2017, Long Beach, California, USA, p. 6000
[19] Wang J, Xiao C, Wang S, *et al.* 2023 *The Journal of Engineering* **9** e12303
[20] Ling Z, Zhang Y and Chen X 2023 *IEEE Transactions on Intelligent Transportation Systems* **24** 5871
[21] Xing Z, Tu S and Xu L 2020 arXiv:2005.06879 [cs.LG]
[22] Luo J, Li C, Fan Q, *et al.* 2022 *Engineering Applications of Artificial Intelligence* **112** 104848
[23] Deudon M, Cournut P, Lacoste A, *et al.* 2018 *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, June, 2018, p. 170
[24] Kool W, Van Hoof H and Welling M 2019 *Attention, learn to solve routing problems! In Proceedings of the 7th International Conference on Learning Representations (ICLR)*, May 6–9, 2019, New Orleans, Louisiana, United States, p. 25
[25] Paulo R d O Costa, Rhuggenaath J, Zhang Y, *et al.* 2020 *In Asian Conference on Machine Learning*, November 18–20, 2020, Bangkok, Thailand, p. 465
[26] Ouyang W, Wang Y, Han S, *et al.* 2021 *IEEE Symposium Series on Computational Intelligence*, December 5–7, 2021, Orlando, FL, USA, p. 1

[27] Vitali T, Mele U J, Gambardella L M, *et al.* 2021 arXiv:2108.00938 [cs.AI]
[28] Parasteh S, Khorram A, Mouhoub M, *et al.* 2022 *IEEE International Conference on Systems, Man, and Cybernetics*, October 9–12, 2022, Prague, Czech Republic, p. 2514
[29] Wu Y, Song W, Cao Z, *et al.* 2021 *IEEE Transactions on Neural Networks and Learning Systems* **33** 5057
[30] Navarro D J, Newell B R and Schulze C 2016 *Cognitive Psychology* **85** 43
[31] Pei M, An H, Liu B, *et al.* 2021 *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **52** 4415
[32] Ghannadi P, Kourehli S S and Mirjalili S 2023 *Frattura ed Integrità Strutturale* **64** 51
[33] Mirjalili S and Lewis A 2016 *Advances in Engineering Software* **95** 51
[34] Zheng J, Yuan T, Xie W, *et al.* 2023 *Sensors* **23** 6463
[35] Benyamin A, Farhad S G and Saeid B 2021 *International Journal of Intelligent Systems* **36** 1270
[36] Sutton R and Barto A 1998 *IEEE Transactions on Neural Networks* **9** 1054
[37] Srivastava N, Hinton G, Krizhevsky A, *et al.* 2014 *The Journal of Machine Learning Research* **15** 1929
[38] Williams R J 1992 *Machine Learning* **8** 229
[39] Al-Hamadani M N A, Fadhel M A, Alzubaidi L, *et al.* 2024 *Sensors* **24** 2461
[40] Chen X and Tian Y 2019 *Advances in Neural Information Processing Systems* **2019** 32
[41] Reinhelt G 2014 {TSPLIB}: a library of sample instances for the TSP (and related problems) from various sources and of various types
[42] Xu M, Pan Q, Muscoloni A, *et al.* 2020 *Nat. Commun.* **11** 2849
[43] Helsgaun K 2017 *Roskilde: Roskilde University* **12** 966
[44] Applegate D, Bixby R, Chvatal V, *et al.* 2006 Concorde TSP solver
[45] Perron L and Furnon V 2019 OR-Tools (version 7.2)
[46] Lei K, Guo P, Wang Y, *et al.* 2022 *Neurocomputing* **508** 79
[47] Wang Q and Tang C 2021 *Knowledge-Based Systems* **233** 107526
[48] Vinyals O, Fortunato M and Jaitly N 2015 *Advances in Neural Information Processing Systems*, December, 2015, Montreal, Canada, p. 2692
[49] Liu H, Zong Z, Li Y, *et al.* 2023 *Applied Soft Computing* **146** 110680
[50] Rožić T, Naletina D and Zając M 2022 *Applied Sciences* **12** 8452
[51] Verschuur J, Salmon N, Hall J, *et al.* 2024 *Environmental Research: Infrastructure and Sustainability* **4** 015001
[52] Rbihou S and Haddouch K 2021 *Fifth International Conference on Intelligent Computing in Data Sciences*, October 20–22, 2021, Fez, Morocco, p. 1
[53] Wang S, Liu M and Chu F 2020 *Journal of Cleaner Production* **249** 119433
[54] Kalatzantonakis P, Sifaleras A and Samaras N 2023 *Expert Systems with Applications* **213** 118812