Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning

Jingyan Sui SUIJINGYAN@ICT.AC.CN
Shizhe Ding DINGSHIZHE15@MAILS.UCAS.AC.CN
Ruizhi Liu LIURUIZHI19S@ICT.AC.CN
Liming Xu XULIMING19@MAILS.UCAS.AC.CN
Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing 100049, China

Dongbo Bu DBU@ICT.AC.CN
Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing 100049, China
Zhongke Big Data Academy, Zhengzhou 450046, Henan, China

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Traveling salesman problem (TSP) is a classical combinatorial optimization problem. As it represents a large number of important practical problems, it has received extensive studies and a great variety of algorithms have been proposed to solve it, including exact and heuristic algorithms. The success of heuristic algorithms relies heavily on the design of powerful heuristic rules, and most of the existing heuristic rules were manually designed by experienced experts to model their insights and observations on TSP instances and solutions. Recent studies have shown an alternative promising design strategy that directly learns heuristic rules from TSP instances without any manual interference. Here, we report an iterative improvement approach (called Neural-3-OPT) that solves TSP through automatically learning effective 3-opt heuristics via deep reinforcement learning. In the proposed approach, we adopt a pointer network to select 3 links from the current tour, and a feature-wise linear modulation network to select an appropriate way to reconnect the segments after removing the selected 3 links. We demonstrate that our approach achieves state-of-the-art performance on both real TSP instances and randomly-generated instances than, to the best of our knowledge, the existing neural network-based approaches.

Keywords: Deep reinforcement learning, Traveling salesman problem, 3-opt, FiLM-Net

1. Introduction

Traveling salesman problem (TSP) is a classical combinatorial optimization problem that can be stated as follows: given a collection of cities with known location coordinates, the goal is to find the shortest tour that visits every city exactly once and finally returns to the starting city. Using graph theory language, a TSP instance can be described as an indirected graph, where a node represents a city, and for each pair of cities, we connect them using a link with weight to represent the distance between them. TSP is the natural and essential formulation of a large number of important practical problems, say, shortest airport routes, optimum power delivery and microchip design. Thus, TSP has received extensive studies and a great variety of algorithms have been proposed to solve TSP.

The algorithms to solve TSP can be divided into two categories, namely, exact algorithms and heuristic algorithms. The exact algorithms, e.g., brute-force enumeration(Gutin and Punnen, 2006), dynamic programming(Held and Karp, 1962) and integer linear programming (Dantzig et al., 1954), could find the exact shortest tour among the given cities; however, these algorithms are inevitably time-consuming due to the NP-hardness essence of TSP. Unlike the exact algorithms finding the optimal solutions, heuristic algorithms aims to find sufficient good solution. For instance, 2-opt algorithm starts from an arbitrary tour and iteratively improves it through selecting 2 links from current tour, removing them, and reconnecting the remaining segments. Compared with exact algorithms, heuristic algorithms are usually much more fast and efficient with only a little sacrifice of solution quality.

The success of heuristic algorithms relies heavily on the design of powerful heuristic rules. Most of the existing heuristic rules were manually designed by experienced experts to model their insights and observations on TSP instances and solutions. This manual-design strategy has its advantage in accurately capturing the intuition and insights of algorithm designer but sometimes suffers from the relatively low generality — an algorithm designer could analyse only a limited number of instances to obtain insights and intuitions; thus, the designed heuristics working well on one type of TSP instances might perform poorly on another type of TSP instances.

Recent studies have shown an alternative promising design strategy that directly learns heuristic rules from TSP instances without any manual interference. For example, d O Costa et al. (2020) proposed an approach (referred to as L-2-OPT hereafter) to learn 2-opt heuristic directly from abundant randomly-generated instances using deep reinforcement learning. Compared with the manual-design strategy, this 'heuristic learning' design strategy generates heuristic rules through analysing much more instances, and thus achieving the potential advantage to significantly increase the generality of the designed heuristics.

In the study, we focus on the design of effective 3-opt heuristic, which selects 3 links, rather than 2 links as performed by 2-opt heuristics, to remove and reconnect for improvement. Previous studies have shown that the power of a 3-opt operation is roughly equivalent to that of three 2-opt operations (Helsgaun, 2000), implying that 3-opt heuristic is more efficient and effective in finding the optimal solution than 2-opt heuristic. It should be pointed out that 3-opt heuristic is not a trivial extension of 2-opt heuristic: for any two links selected by a 2-opt heuristic, we have only one way to reconnect them. In contrast, after selecting 3 links from a tentative tour, we have a total of 7 ways to reconnect them and a 3-opt heuristic should determine the best reconnecting type, which makes the design of effective 3-opt heuristics extremely challenging.

Here, we report an approach (called Neural-3-OPT) that solves TSP through automatically learning effective 3-opt heuristics. Our approach consists of the following three main components: *i*) a graph convolution network and a bidirectional long short-term memory network to encode nodes of the input TSP instance. To reduce computation cost, the encoder uses a k-nearest neighbor module to limit the encoding range within k nearest neighbors of each node only. *ii*) a pointer network to select 3 links from the current tour, and *iii*) a feature-wise linear modulation network to select an appropriate type to reconnect the segments after removing the selected 3 links. Using both real and randomly-generated TSP instances, we demonstrate that our approach achieves state-of-the-art performance than, to the best of our knowledge, the existing neural network-based approaches.

2. Related work

The algorithms to solve TSP include exact algorithms and heuristic algorithms. The exact algorithms guarantee to obtain the optimal solutions for the given instances by using integer linear programming (Dantzig et al., 1954), branch-and-bound (Little et al., 1963) or other techniques (Laporte, 1992). Due to the NP-hardness of TSP, the exact algorithms are time-consuming and thus cannot solve large instances within reasonable time. In the study, we use Concorde (Applegate et al., 2006), an exact algorithm, to calculate optimal solutions to benchmark the heuristic algorithms.

Unlike the exact algorithms attempting to generate the optimal solutions, the heuristic algorithms aim to generate sufficiently good solutions within reasonable time. For a given TSP instance, some heuristic algorithms construct tour from the very scratch. For example, the nearest-neighbor algorithm starts from an arbitrarily-selected node and keeps moving to the nearest neighbor until visiting all nodes, thus acquiring a reasonably short tour (Flood, 1956; Robacker, 1955). Alternatively, some heuristic algorithms adopt the "step-by-step improvement" strategy, i.e., starting from an initial complete tour that covers all nodes, and iteratively shorten the tour through replacing some links with others (Lin, 1965; Johnson and McGeoch, 2002; Helsgaun, 2017). The representative algorithms using this strategy include k-opt (Lin, 1965), local search(Johnson and McGeoch, 2002), Lin-Kernighan-Helsgaun-3 (LKH-3) (Helsgaun, 2017) and Google OR-Tools.

The success of heuristic algorithms relies heavily on the design of appropriate heuristic rule, i.e., how to select adding nodes in the "construction from the very scratch" strategy, and how to select removing and adding links in the "step-by-step improvement" strategy. Previous studies usually use the hand-crafted heuristics designed by experienced experts to describe their insights into TSP instances and optimal solutions. Recent studies have shown a promising design strategy that directly learns heuristics from data using deep learning (DL) or deep reinforcement learning (DRL) (Vinyals et al., 2015; Bello et al., 2016; Dai et al., 2017; Kool et al., 2018; Joshi et al., 2019; Applegate et al., 2006; Lu et al., 2019; Wu et al., 2019; Zheng et al., 2020; d O Costa et al., 2020), which are summarized as follows.

- For the algorithms to solve TSP using the "construction from very scratch" strategy, the heuristics to select appropriate nodes can be learned from TSP instances using supervised learning(Vinyals et al., 2015; Joshi et al., 2019) or reinforcement learning (RL) (Bello et al., 2016; Dai et al., 2017; Kool et al., 2018; Deudon et al., 2018). For example, pointer network(Vinyals et al., 2015) uses an encoder-decoder model together with attention mechanism to predict the selection probability distribution over nodes. The pointer network can be trained using both supervised learning and reinforcement learning techniques (Bello et al., 2016). By using stacked attention layers, this framework achieves significant performance improvement especially when enhanced with rollout baseline(Kool et al., 2018). In accordance with the graph essence of TSP instances, graph neural network was also used to encode the features of nodes' neighbors(Dai et al., 2017; Joshi et al., 2019).

- For the algorithms using the "step-by-step improvement" strategy, heuristic rules to select removing/adding links can be learned using DRL (Lu et al., 2019; Wu et al., 2019; Zheng et al., 2020; d O Costa et al., 2020). In these algorithms, the selected links are represented as adjacent matrix (Wu et al., 2019) or node pairs (d O Costa et al., 2020). This DRL can also be combined with the classical hand-crafted heuristic rule like LKH (Zheng et al., 2020).

Previous studies has shown that the learned 2-opt heuristic rules have considerable performance and generality to larger TSP instances. The possibility of extending to k-opt has also been claimed; however, a systematic implementation of this idea has not been proposed. In this study, we present an investigation of learning 3-opt heuristics using neural networks.

3. Background

The k-opt heuristics are widely used in the "step-by-step improvement" strategy to solve TSP, which remove k links from the current tour and reconnect the thus-acquired segments to yield an improved tour. As shown in Figure 1, when applying 2-opt on a tour, we have only one reconnecting type. In contrast, when applying 3-opt, we have a total of 7 reconnecting types. It should be pointed out that the 7 reconnecting types can be clustered into 3 categories according to permutation equivalence.

The key points of a k-opt heuristic are selecting appropriate k links from the current tour and selecting an appropriate reconnecting type after removing the selected links. A straightforward idea is selecting the links and reconnecting type that lead to the greatest decrease of tour length. However, this greedy strategy often generates solutions with significantly poor performance. In the study, we present a model to learn a strategy to select links and determine the appropriate reconnecting type for the selected links.

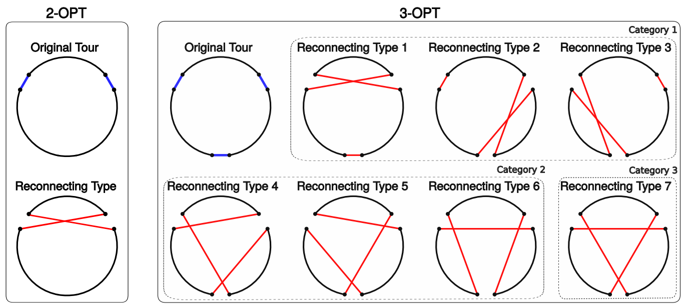


Figure 1: Illustration of 2-opt and 3-opt heuristics used in the "step-by-step improvement" strategy to solve TSP. The links in blue represent the links selected to be removed from the current tour. The removement of k links will always lead to k segments (shown as edges in black). After adding new links (shown in red), we acquire a new tour. For 2-opt move, we have only one reconnecting type. In contrast, for the 3-opt move, we have a total of 7 reconnecting types, which can be clustered into 3 categories according to permutation equivalence of them.

4. Method

4.1. Overview of our approach

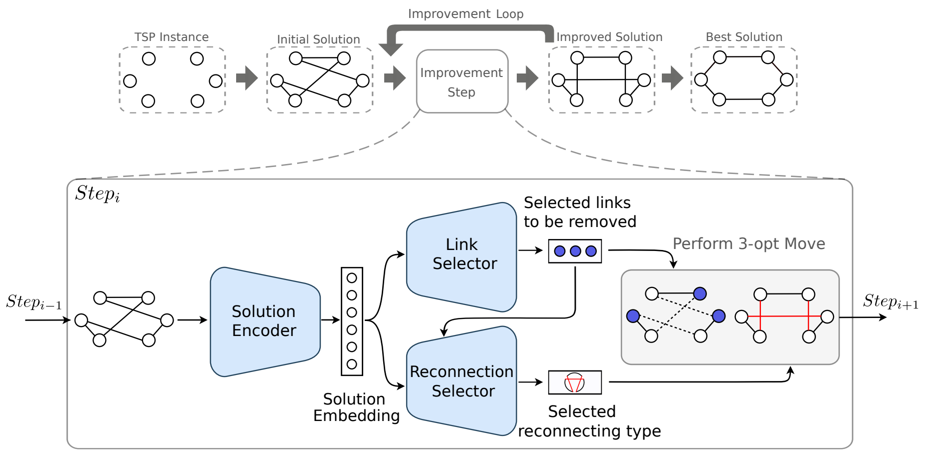


Figure 2: Overview of our Neural-3-OPT approach to TSP. Starting from a randomly-generated initial tour of a given TSP instance, Neural-3-OPT keeps improving it by applying 3-opt heuristic. At each improvement step, an encoder calculate embedding for the current solution, which is then fed into two decoders, including a link-selector and a reconnection-selector, to select 3 links to be removed and the appropriate reconnecting type.

Figure 2 shows the basic idea of our Neural-3-OPT approach to TSP. For a given TSP instance, Neural-3-OPT starts from an initial tour that is constructed randomly, and keeps improving it by applying 3-opt heuristic. At each improvement step, Neural-3-OPT uses an encoder to calculate embedding for the current solution. The embedding is then fed into two decoders, including a link-selector to report 3 links to be removed from the current solution, and a reconnection-selector to determine the appropriate reconnecting type.

In the following subsections, we first describe the Markov decision process (MDP) formulation of the improvement strategy to solve TSP. Next, we provide details of the neural network model used by Neural-3-OPT, including its architecture and training process to optimize its parameters.

4.2. Markov decision process formulation of the improvement strategy

Here, we adopt the MDP formulation similar to that used in Wu et al. (2019) and d O Costa et al. (2020) with a slight change of the action definition. The formulation are described in details as follows.

State The state at the t-th improvement step, denoted as $\bar{S}_t = (S_t, S'_t)$, is a tuple that contains two tours, the current solution S_t, and the best solution so far S'_t.

Action At the t-th improvement step, an action to change the current state $\bar{S}_t$ is represented as a tuple $A_t = (a_0, a_1, a_2, T_{3\text{-}opt})$, where $a_0, a_1, a_2 \in S_t$ denotes the selected links to be removed from the current tour, and T_{3-opt} denotes the selected reconnecting type to reconnect the segments after removing the selected 3 links.

Transaction The transaction $T(\bar{S}_t, A_t) \to \bar{S}_{t+1}$ yields a new state after applying the action A_t onto the current state $\bar{S}_t$, which describes a 3-opt move on the current tour.

Reward Reward is defined as $R_t = L(S'_t) - \min(L(S'_t), L(S_{t+1}))$, i.e., the improvement of the best tour so far at the t-th improvement step, where $L(.)$ represents length of a tour.

Returns We define cumulative reward from time step t to the pre-defined episode length T with discount factor $\gamma \in (0, 1]$ as $G_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} R_{t'}$.

4.3. Neural network architecture used by Neural-3-OPT

Similar to d O Costa et al. (2020), our Neural-3-OPT approach also uses the general encoder-decoder architecture together with actor-critic training process. It should be pointed out the two main differences between our approach and L-2-OPT:

- Considering that there is only one reconnecting type in 2-opt heuristic, L-2-OPT uses only one decoder in policy network to select links, without selecting reconnecting type. In contrast, to handle the issue of seven reconnecting types in 3-opt heuristic, we use two neural networks as two decoders, including a link selector and a reconnection selector to select the links to be removed and the appropriate reconnecting type at the same improvement step.
- In order to reduce the computation cost in the encoding stage, we use a sparse K-nearest-neighbor graph (G_K) in encoder to limit the encoding range within k nearest neighbors of each node only, rather than using the complete graph, which has proven to provide more effective encoding and encourage convergence in training process.

The details of the neural network architecture are described as follows.

4.3.1. Encoder

The encoder consists of two neural modules: a sparse graph convolution network (K-GCN) and a bidirectional long short-term memory network (bi-LSTM), which takes a tour as input and returns the embedding of each node and the entire tour. Given a tour with n nodes $\{x_i\}_n$, a sparse G_K is generated to record K nearest neighbors for each node. Then G_K is fed into N K-GCN layers and the graph node embedding $h_i^{(l)}$ is calculated as :

$$h_i^{(l+1)} = h_i^{(l)} + \mathrm{ReLU}\Big(\sum_{j \in \mathcal{N}(i)} e_{ij} (W^{(l)} h_j^{(l)} + b^{(l)})\Big), \tag{1}$$

where h_0 is the initial d-dimensional node embedding calculated by a linear projection from x_i to d-dimensional space, e_{ij} is the normalized edge weight of G_K defined as reciprocal of edge length, and $W^{(l)} \in \mathbb{R}^{d \times d}$, $b^{(l)} \in \mathbb{R}^d$ are the l-th K-GCN layer's parameters.

Similar to d O Costa et al. (2020), a bi-LSTM module is used to encode features of tour sequence with the resulting node embedding from K-GCN. We represent the sequential node embedding o_i and the sequential tour embedding h_n as

$$o_i = \tanh\left((W_f h_i^{\rightarrow} + b_f) + (W_b h_i^{\leftarrow} + b_b)\right) \tag{2}$$

$$h_n = h_n^{\rightarrow} + h_n^{\leftarrow} \tag{3}$$

where $h_i^{\rightarrow}, h_i^{\leftarrow} \in \mathbb{R}^d$ are the hidden vectors of bi-LSTM, and $W_f, W_b \in \mathbb{R}^{d\times d}$, $b_f, b_b \in \mathbb{R}^d$ are parameters.

As formulated before, state $\bar{S}_t = (S_t, S'_t)$ consists of two tours: current tour S_t and the best tour so far S'_t, therefore we use two independent encoder to calculate their embedding. Similarly to h_n, we can represent $h'_n \in \mathbb{R}^d$ as the the sequential tour embedding of tour S'_t.

4.3.2. Policy Decoders

Policy decoders $\pi_{\theta,\phi}$ consist of two neural modules: a link selector π_θ and a reconnection selector π_ϕ, shown in Figure 3. Taken tour embedding h_n and h'_n from encoders as input, the link selector outputs three links sequentially, and the reconnection selector outputs an appropriate reconnecting type.

We use chain rule to factorize the probability of a 3-opt move as

$$\begin{aligned}\pi_{\theta,\phi}(A \mid \bar{S}) &= \pi_\theta\left(a_{<3} \mid \bar{S}\right) \cdot \pi_\phi\left(T_{3\text{-}opt} \mid a_{<3}, \bar{S}\right) &(4)\\ &= \prod_{i=1}^{3} p_\theta\left(a_i \mid a_{<i}, \bar{S}\right) \cdot p_\phi\left(T_{3\text{-}opt} \mid a_{<3}, \bar{S}\right), &(5)\end{aligned}$$

where a_i is the i-th selected link, $a_{<i}$ represents the collection of previously selected links and $T_{3\text{-}opt}$ is the selected reconnecting type.

In link selector π_θ, a pointing mechanism, similar to d O Costa et al. (2020), is used to output links (a_0, a_1, a_2) sequentially based on three predicted distributions over links given the node embedding $\{o_i\}_n$ and state representations i.e. query vectors (q_0, q_1, q_2), which are calculated as

$$\begin{aligned} q_0 &= \left(W_s h_n + b_s \| W_{s'} h'_n + b_{s'}\right) + max(h_i^{(N)}) &(6)\\ q_i &= \tanh\left((W_q q_{i-1} + b_q) + (W_o o_{i-1} + b_o)\right). &(7)\end{aligned}$$

where $W_s, W_{s'} \in \mathbb{R}^{\frac{d}{2}\times d}, W_q, W_o \in \mathbb{R}^{d\times d}$ and $b_s, b_{s'} \in \mathbb{R}^{\frac{d}{2}}, b_q, b_o \in \mathbb{R}^d$ are parameters. The pointing mechanism is parameterized as

$$p_\theta\left(a_i \mid a_{<i}, \bar{S}\right) = \operatorname{softmax}\left(C \tanh\left(u^i\right)\right), \tag{8}$$

where

$$u_j^i = \begin{cases} v^T \tanh\left(K o_j + Q q_i\right) & \text{if } j > a_{i-1} \\ -\infty, & \text{otherwise} \end{cases}, \tag{9}$$

with $K, Q \in \mathbb{R}^{d\times d}$ and $v \in \mathbb{R}^d$ as learned attention parameters, and C is a super parameter.

In reconnection selector π_ϕ, we treat selecting reconnecting type as a classification problem, and we adpot three alternative types, which come form each category of 3-opt respectively. We adopt Feature-wise Linear Modulation Network (FiLM-Net) (Perez et al., 2018; Gupta et al., 2020) to modulate the current solution features with selected links features as conditioning information to output the appropriate reconnecting type based on predicted distributions over three alternative types. FiLM-Net is a strong conditioning method that combines additive and multiplicative interactions into a conditional affine transformation, which is more efficient than single addition or multiplication for integrating multiple sources of information.

We adopt L FiLM-Net layers, and each layer learns two functions $f_\phi^{(l)}$ and $g_\phi^{(l)}$ ($l \in \{1, \ldots, L\}$) based on neural network, which maps the sum embedding $h^{(c)}$ of three selected links to FiLM parameters $\gamma^{(l)}, \beta^{(l)} \in \mathbb{R}^d$:

$$\gamma^{(l)} = f_\phi^{(l)}(h^{(c)}), \quad \beta^{(l)} = g_\phi^{(l)}(h^{(c)}), \tag{10}$$

then $\gamma^{(l)}$ and $\beta^{(l)}$ are used to modulate input h_m as:

$$h_m^{(l+1)} = \text{FiLM}^{(l)}\left(h_m^{(l)} \mid \gamma^{(l)}, \beta^{(l)}\right) = \gamma^{(l)} \odot h_m^{(l)} + \beta^{(l)}, \tag{11}$$

where input of the first layer $h_m^{(0)}$ refers to tour embedding h_n. Then the distribution $p_\phi\left(T_{3\text{-}opt} \mid a_{<3}, \bar{S}\right)$ on reconnecting types is generated by a softmax layer applying on the result embedding $h_m^{(L)}$ of FiLM-Net:

$$p_\phi\left(T_{3\text{-}opt} \mid a_{<3}, \bar{S}\right) = \text{softmax}\left(W_m h_m^{(L)} + b_m\right), \tag{12}$$

where $W_m \in \mathbb{R}^{3\times d}$, and $b_m \in \mathbb{R}^3$ are parameters.

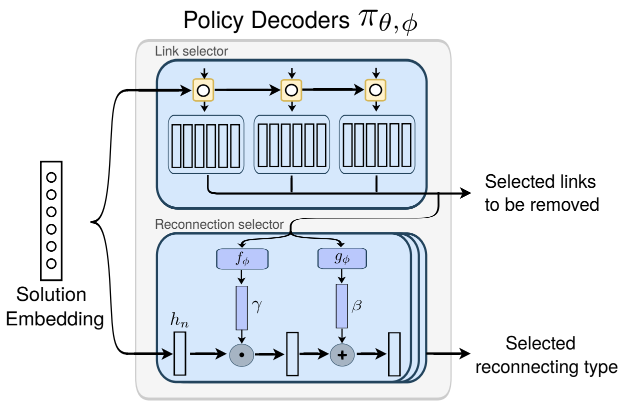

Policy Decoders $\pi_{\theta,\phi}$
Link selector
Selected links to be removed
Reconnection selector
f_ϕ
g_ϕ
γ
β
h_n
Solution Embedding
Selected reconnecting type

Figure 3: Architecture of policy decoders. Policy decoders consist of a Pointer-Net as link selector and a FiLM-Net as reconnection selector. With tour embedding from encoders as input, the link selector selects three links to be removed. With embedding from encoders as input and three selected links as conditioning information, the reconnection selector selects an appropriate reconnecting type.

4.3.3. Value decoder

Similar to d O Costa et al. (2020), a value decoder takes graph representation from S and tour representation from S and S', then outputs a real value to estimate current state value for Actor-Critic policy optimization, formulated as:

$$V_{\psi}(\bar{S}) = W_r \,\mathrm{ReLU}\left(W_z\left(\frac{1}{n}\sum_{i=1}^{n} h_i^{(L)} + \left(W_v h_n + b_v \| W_{v'} h'_n + b_{v'}\right)\right) + b_z\right) + b_r, \tag{13}$$

where $W_v, W_{v'} \in \mathbb{R}^{\frac{d}{2}\times d}, W_z \in \mathbb{R}^{d\times d}, W_r \in \mathbb{R}^{1\times d}$, $b_v, b_{v'} \in \mathbb{R}^{\frac{d}{2}}, b_z \in \mathbb{R}^{d}$, and $b_r \in \mathbb{R}$.

4.4. Actor-Critic policy optimization

We resort to the similar policy gradient optimization with d O Costa et al. (2020), except adding an extra entropy bonus corresponding to the reconnection selector.

We define objective function of policy network as the expected returns given a state $\bar{S}$: $J(\theta, \phi \mid \bar{S}) = \mathbb{E}_{\pi_{\theta,\phi}}\left[G_t \mid \bar{S}\right]$, and our objective is to maximize it.

We optimize $J(\theta, \phi \mid \bar{S})$ by gradient descent, using Actor-Critic:

$$\nabla J(\theta, \phi) \approx \frac{1}{B}\frac{1}{T}\left[\sum_{b=1}^{B}\sum_{t=0}^{T-1} \nabla_{\theta,\phi} \log \pi_{\theta,\phi}\left(A_t^b \mid \bar{S}_t^b\right) \mathcal{A}_t^b\right]. \tag{14}$$

with $\mathcal{A}_t^b = G_t^b - V_{\psi}\left(\bar{S}_t^b\right)$ as advantage. We add an entropy bonus

$$H(\theta, \phi) = \frac{1}{B}\sum_{b=1}^{B}\sum_{t=0}^{T-1} H\left(\pi_{\theta,\phi}\left(\cdot \mid \bar{S}_t^b\right)\right) \tag{15}$$

with $H\left(\pi_{\theta,\phi}\left(\cdot \mid \bar{S}_t^b\right)\right) = -\mathbb{E}_{\pi_{\theta,\phi}}\left[\log \pi_{\theta,\phi}\left(\cdot \mid \bar{S}_t^b\right)\right]$.

We define objective of value network as:

$$\mathcal{L}(\psi) = \frac{1}{B}\frac{1}{T}\left[\sum_{b=1}^{B}\sum_{t=0}^{T-1} \|G_t^b - V_{\psi}\left(\bar{S}_t^b\right)\right) \|_2^2\right] \tag{16}$$

We use ADAM (Kingma et al., 2015) optimizer. The complete algorithm is shown in Algorithm 1.

Algorithm 1 Training process of Neural-3-OPT

```
1: Input: number of epochs $E$; number of mini-batches $N_B$; batch size B; neighbor size K; step limit $\mathbb{T}$; length of episodes $T_e$; learning rate $\lambda$;
2: Initialize parameters $\theta$, $\phi$ and $\psi$ of policy network $\pi_{\theta,\phi}$ and critic network $V_{\psi}$;
3: for epoch $= 1, \ldots,$ do
4:     $T \leftarrow T_e$
5:     for batch $= 1, \ldots, N_B$ do
6:         $t \leftarrow 0$
7:         Initialize random $\bar{S}_0^b$, $\forall b \in \{1, \ldots, B\}$
8:         while $t < \mathbb{T}$ do
9:             $g_{\theta,\phi} \leftarrow \frac{1}{B}\left[\frac{1}{T}\sum_{b=1}^{B}\sum_{i=0}^{T-1} \nabla_{\theta,\phi} \log \pi_{\theta}\left(a_{<3} \mid \bar{S}_i^b\right) \pi_{\phi}\left(T_{3\text{-}opt} \mid a_{<3}, \bar{S}_i^b\right) \mathcal{A}_i^b + \beta_H \nabla_{\theta,\phi} H\left(\pi_{\theta,\phi}\right)\right]$
10:            $g_{\psi} \leftarrow \frac{1}{BT}\left[\beta_V \sum_{b=1}^{B}\sum_{i=0}^{T-1} \nabla_{\psi} \|G_i^b - V_{\psi}\left(\bar{S}_i^b\right)\right) \|_2^2\right]$
11:            $\theta, \phi, \psi \leftarrow \mathrm{ADAM}\left(\lambda, -g_{\theta,\phi}, g_{\psi}\right)$
12:        end while
13:    end for
14: end for
```

Table 1: Quality of the tours reported by our approach and the existing approaches. *Type*: **SL**: Supervised Learning, **RL**: Reinforcement Learning, **S**: Sampling, **G**: Greedy, **B**: Beam Search, **BS**: **B** and Shortest Tour and **T**: 2-opt Local Search. *Length*: length of the output tour. *Gap*: optimality gap. *Time*: the total time to solve all 10,000 instances in dataset reported in d O Costa et al. (2020) and ours.

	Method	Type	TSP20			TSP50			TSP100		
			Length	Gap	Time	Length	Gap	Time	Length	Gap	Time
	Concorde (Applegate et al., 2006)	Solver	3.84	0.00%	2m	5.70	0.00%	13m	7.76	0.00%	59m
Heuristic	OR-Tools	S	3.85	0.37%		5.80	1.83%		7.99	2.90%	
	Nearest Insertion	G	4.33	12.91%	1s	6.78	19.03%	2s	9.46	21.82%	6s
	Random Insertion	G	4.00	4.36%	0s	6.13	7.65%	1s	8.52	9.69%	3s
	Farthest Insertion	G	3.93	2.36%	1s	6.01	5.53%	2s	8.35	7.59%	7s
Const.+Greedy	PreNet(Vinyals et al., 2015)	SL	3.88	1.15%		7.66	34.48%			-	
	GCN (Joshi et al., 2019)	SL	3.86	0.60%	6s	5.87	3.1%	55s	8.41	8.38%	6m
	PtrNet (Bello et al., 2016)	RL	3.89	1.42%		5.95	4.46%		8.30	6.90%	
	S2V (Dai et al., 2017)	RL	3.89	1.42%		5.99	5.16%		8.31	7.03%	
	GAT (Deudon et al., 2018)	RL,T	3.85	0.42%	4m	5.85	2.77%	26m	8.17	5.21%	3h
	GAT Kool et al. (2018)	RL	3.85	0.34%	0s	5.80	1.76%	2s	8.12	4.53%	6s
Const.+Search	GCN (Joshi et al., 2019)	SL,B	3.84	0.10%	20s	5.71	0.26%	2m	7.92	2.11%	10m
	GCN (Joshi et al., 2019)	SL,BS	3.84	0.01%	12m	5.70	**0.01**%	18m	7.87	1.39%	40m
	PtrNet (Bello et al., 2016)	RL,S		-		5.75	0.95%		8.00	3.03%	
	GAT (Deudon et al., 2018)	RL,S	3.84	0.11%	5m	5.77	1.28%	17m	8.75	12.70%	56m
	GAT (Deudon et al., 2018)	RL,S,T	3.84	0.09%	6m	5.75	1.00%	32m	8.12	4.64%	5h
	GAT (1280) (Kool et al., 2018)	RL,S	3.84	0.08%	5m	5.73	0.52%	24m	7.94	2.26%	1h
Impr.+Sampling	GAT-T (1000) (Wu et al., 2019)	RL	3.84	0.03%	12m	5.75	0.83%	16m	8.01	3.24%	25m
	GAT-T (3000) (Wu et al., 2019)	RL	3.84	0.00%	39m	5.72	0.34%	45m	7.91	1.85%	1h
	GAT-T (5000) (Wu et al., 2019)	RL	3.84	0.00%	1h	5.71	0.20%	1h	7.87	1.42%	2h
	L-2-OPT (500) (d O Costa et al., 2020)	RL	3.84	0.01%	8m	5.72	0.36%	11m	7.91	1.84%	15m
	L-2-OPT (1000) (d O Costa et al., 2020)	RL	3.84	0.00%	16m	5.71	0.21%	21m	7.86	1.26%	29m
	L-2-OPT (2000) (d O Costa et al., 2020)	RL	3.84	0.00%	31m	5.70	0.12%	41m	7.83	0.87%	59m
	Ours (500)	RL	3.84	0.04%	8m	5.71	0.24%	12m	7.89	1.65%	20m
	Ours (1000)	RL	3.84	0.01%	16m	5.70	0.12%	25m	7.85	1.06%	39m
	Ours (2000)	RL	**3.84**	**0.00**%	32m	**5.70**	**0.08**%	48m	**7.82**	**0.74**%	80m

5. Experiments and results

5.1. Experiment setting

We evaluated Neural-3-OPT and compared it with the existing approaches on four benchmark datasets, including TSP20, TSP50, TSP100 and TSP200, which contain 10000 instances with 20, 50, 100 and 200 nodes respectively. The nodes in these datasets are randomly drawn from the unit square $[0, 1]^2$. For the sake of comparison, we used the identical datasets TSP20, TSP50 and TSP100 to those used by Kool et al. (2018) and d O Costa et al. (2020). In addition, we also used hyperparameters identical to d O Costa et al. (2020). Specifically, we set batch size $B = 512$, vector dimension $d = 128$, layer number $N, L = 3$, loss weights $\beta_v = 0.5$, $\beta_H = 0.005$, and neighbor size $K = 8$ for G_K.

We run 200 epochs for TSP20 dataset, 300 epochs for TSP50 and TSP100 dataset, and 500 epochs for TSP200 dataset. The episode length are $T_1 = 8$, $T_{100} = 10$, $T_{150} = 20$ for TSP20, and $T_1 = 8$, $T_{100} = 10$, $T_{200} = 20$ for TSP50, TSP100 and TSP200.

To make the comparison fair, we implement evaluation experiments by running our approach and L-2-OPT in the same platform (CPU: Intel Xeon; GPU: RTX 3090).

Table 2: Solving the real-world instances recorded in TSPlib using OR-Tools, L-2-OPT and Neural-3-OPT. Here, Neural-3-OPT is trained using randomly-generated instances.

Instance	Opt.	OR-Tools	L-2-OPT	Ours
eil51	426	439	427	436
berlin52	7,542	7,944	7,974	**7,572**
pr76	108,159	110,948	111,085	**108,277**
rd100	7,910	8,221	7,944	8,077
eil101	629	650	635	640
lin105	14,379	15,363	16,156	**15,229**
ch130	6,110	6,329	6,175	6,213
pr144	58,537	59,286	61,207	**60,851**
ts225	126,643	127,763	127,731	129,381
a280	2,579	2,742	2,898	**2,808**
Avg. Opt. Gap	0.00%	3.79%	4.56%	**2.93%**

5.2. Solving randomly-generated TSP instances using the learned heuristics

For each TSP instance in the test datasets TSP20, TSP50 and TSP100, we run our Neural-3-OPT approach, L-2-OPT and Concorde to yield tours and running times. To evaluate the output tours, we calculate their lengths and compare with the optimal tour reported by Concorde. The difference between the length of an output tour and that of the optimal tour is denoted as "optimality gap".

As shown in Table 1, our approach Neural-3-OPT finds the optimal tour for all the 10,000 instances in TSP20. For the instances in TSP50 and TSP100, our approach achieves optimal gap as low as of 0.08% and 0.74%, respectively. In contrast, the L-2-OPT approach achieves large optimality gap (0.12% on TSP50, and 0.87% on TSP100), although it finds the optimal tours for the instances in TSP20. These results clearly suggest that our approach outperforms the existing heuristics and neural network based approaches.

On the TSP20 dataset, our approach uses roughly the same running time as L-2-OPT. On the TSP50 and TSP100 datasets, our approach uses about 15% and 34% more time than L-2-OPT. Although our approach is slower than L-2-OPT, the running time is acceptable.

5.3. Solving the real-world TSP instances using the learned heuristics

It is an interesting question that whether the learned heuristics from randomly-generated TSP instances can be used to solve real-world instances. To answer this question, we use the learned policy trained on TSP100 to solve the real-world instances recorded in TSPlib(Reinelt, 1991).

As shown in Table 2, our approach achieves an average optimality gap of 2.93%, lower than L-2-OPT (4.56%) and OR-Tools (3.79%). More specifically, on half of total 10 instances, our approach outperforms L-2-OPT, and on 7 instances, our approach outperforms OR-Tools. In the case of berlin52, OR-Tools and L-2-OPT report tours with length of 7944 and 7974, respectively. In contrast, our approach Neural-3-OPT reports a tour with length as low as of 7572.

In summary, these results suggest that the heuristics learned from randomly-generated instances could also facilitate solving the real-world instances.

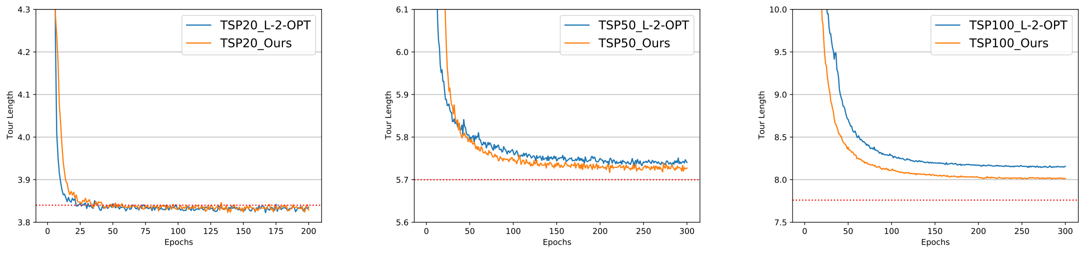
Table 3: Running time of Neural-3-OPT with and without K-GCN. When K-GCN is turned off, we use the original GCN instead.

K-GCN	TSP20	TSP50	TSP100
ON ($K = 8$)	13m	18m	27m
OFF	13m	20m	37m

5.4. Comparing the learned 2-opt and 3-opt heuristics

We further perform a deep examination on the training process of L-2-OPT and Neural-3-OPT approaches. As shown in Figure 4, these two approaches exhibit roughly the same behaviours on TSP20. However, for TSP50 and TSP100 with larger instances, Neural-3-OPT converges much faster than L-2-OPT. This result demonstrates that compared with 2-opt, 3-opt is superior in finding better solution.

Figure 4: Comparing the heuristics learned by Neural-3-OPT and L-2-OPT.



5.5. Ablation study of Neural-3-OPT

The main modules of our approach Neural-3-OPT include K-GCN for encoding and FiLM for merging features. To examine the contributions of these modules, we perform ablation study through turning off these modules and comparing with the full version of Neural-3-OPT. When turning off FiLM and K-GCN, we use multiple-layer perception and GCN instead, respectively.

As shown in Table 4, when using FiLM module, Neural-3-OPT achieves an average tour length of 7.82 on TSP100, which is much shorter than that when FiLM is turned off (8.31 and 8.36). The optimality gap is significantly reduced when using FiLM module.

We also observed that when using K-GCN, the optimal gap of Neural-3-OPT is only slightly worse than that when GCN is used (0.74% vs. 0.73%). However, the running time is considerably reduced from 37m to 27m for TSP100 when using K-GCN (Table 3). In Table 3, we run our policy on 2000 instances because of memory limitation.

As shown in Table 3, encoding each node with its K neighbors takes less time than encoding with all neighbors in graph, no matter on TSP20, TSP50 and TSP100, which illustrates that it can reduce calculations with using K-GCN.

We further examine the performance of Neural-3-OPT when using K-GCN with different setting of k. As shown in Table 5, Neural-3-OPT becomes much faster when using smaller

Table 4: Ablation study of Neural-3-OPT. When FiLM and K-GCN are turned off, we instead use multiple-layer perception and GCN, respectively.

FiLM	K-GCN	TSP50		TSP100	
		Length	Gap	Length	Gap
OFF	OFF	5.70	0.14%	8.31	7.05%
OFF	ON	5.71	0.20%	8.36	7.61%
ON	OFF	5.70	0.08%	7.82	0.73%
ON	ON	5.70	0.08%	7.82	0.74%

Table 5: Performance of Neural-3-OPT when using K-GCN with different K.

K	TSP50			TSP100		
	Length	Gap	Time	Length	Gap	Time
4	5.70	0.14%	47m	8.37	7.77%	74m
8	**5.70**	**0.08%**	48m	**7.82**	**0.74%**	80m
16	5.70	0.15%	52m	7.87	1.13%	83m

K. In addition, when setting $K = 4$, the optimality gap is 0.14% and 7.77% for TSP50 and TSP100, respectively. The performance improves when setting $K = 8$ (optimality gap: 0.08% and 0.74% for TSP50 and TSP100, respectively) but drops when setting $K = 16$.

Together, these results demonstrate that the FiLM module can significantly increases tour quality whereas the K-GCN module greatly reduces the running time of Neural-3-OPT with only a slight sacrifice of its performance.

5.6. Comparing Neural-3-OPT with handcrafted strategies for selecting reconnecting types

As shown in Figure 1, after removing 3 links, we have a total of 7 reconnecting types to reconnect the thus-generated segments. Neural-3-OPT uses deep learning to select an appropriate reconnecting type from them. Thus, it is interesting to compare with handcrafted strategies to select reconnecting type, including random selection, fixed selection of the first reconnecting type, and greedy selection.

As shown in Table 6, Neural-3-OPT achieves an optimality gap of 0.74%, which is much smaller than random selection (4.26%), fixed selection (2.04%), and greedy selection (9.97%). In addition, when compressing the 7 reconnecting types into 3 categories according to permutation equivalence, Neural-3-OPT improve further.

These results clearly demonstrates the power of neural network in selecting appropriate reconnecting types for 3-opt heuristics.

5.7. Generalization ability analysis

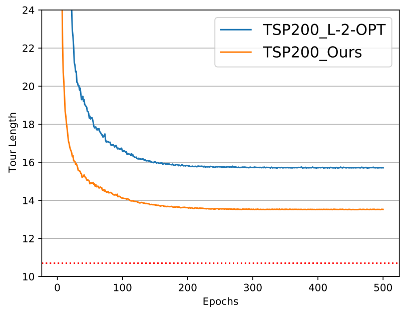
Both Neural-3-OPT and L-2-OPT learn heuristics from TSP instances; thus, it is interesting to investigate the generability of the learned heuristics, i.e., whether the heuristics learned from small size TSP instances can be used to solve large instances.

As shown in Table 7, our method outperforms L-2-OPT on TSP200. As for solving larger scale problems with model trained on smaller scale instances, our method significantly

Table 6: The performance of different strategies to select a reconnecting type.

Strategy to select a reconnecting type	TSP50			TSP100		
	Length	Gap	Time	Length	Gap	Time
Random selection	5.72	0.33%	48m	8.04	4.26%	78m
Fixed selection of type 1	5.71	0.18 %	49m	7.92	2.04%	79m
Greedy selection	5.87	2.99 %	167m	8.54	9.97%	296m
Neural-3-OPT (select from 7 types)	5.70	0.08%	50m	7.83	0.90%	81m
Neural-3-OPT (select from 3 categories)	**5.70**	**0.08**%	48m	**7.82**	**0.74**%	80m

Figure 5: Convergence speed of Neural-3-OPT and L-2-OPT on TSP200. Here, both Neural-3-OPT and L-2-OPT were trained on TSP200.



outperforms L-2-OPT when using TSP50-model on TSP200. When using TSP100-model on TSP200, our method is worse than L-2-OPT. For TSP200, as shown in Figure 5, our method converges faster and achieves a better result than L-2-OPT.

6. Conclusion

In the study, we present an approach to solve TSP using 3-opt heuristics learned from TSP instances. Our approach learns how to select the removing links and how to connect the thus-generated segments as well. Experimental results suggest that the learned heuristics outperform the hand-crafted heuristics. The basic idea shown in the study can be extended to design 4-opt, 5-opt and more complicated k-opt moves without significant changes of the framework.

Table 7: Solving instances in TSP200 using models trained on TSP50/100/200.

Method	Using model trained on TSP200		Using model trained on TSP100		Using model trained on TSP50	
	Length	Gap	Length	Gap	Length	Gap
L-2-OPT	11.63	8.76 %	11.00	2.80 %	11.53	7.82%
Ours	**11.45**	**7.04** %	11.06	3.40 %	**11.45**	**7.02**%

Acknowledgments

We would like to thank the National Key Research and Development Program of China (2020YFA0907000), and the National Natural Science Foundation of China (31671369, 31770775, 62072435) for providing financial supports for this study and publication charges.

References

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver, 2006.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning. In *Asian Conference on Machine Learning*, pages 465–480. PMLR, 2020.

Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

Merrill M Flood. The traveling-salesman problem. *Operations research*, 4(1):61–75, 1956.

Prateek Gupta, Maxime Gasse, Elias B Khalil, M Pawan Kumar, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *arXiv preprint arXiv:2006.15212*, 2020.

Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210, 1962.

Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. 2017.

DS Johnson and LA McGeoch. The traveling salesman problem and Its Variations, Combinatorial Optimization, vol. 12, chap. experimental analysis of heuristics for the STSP, 2002.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

DP Kingma, LJ Ba, et al. Adam: A Method for Stochastic Optimization. 2015.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2019.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *AAAI*, 2018.

Gerhard Reinelt. TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

John T Robacker. Some experiments on the traveling-salesman problem. Technical report, RAND CORP SANTA MONICA CA, 1955.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning Improvement Heuristics for Solving Routing Problems. *arXiv e-prints*, pages arXiv–1912, 2019.

Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, and Chu-min Li. Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. *arXiv preprint arXiv:2012.04461*, 2020.