Reinforcement Learning Approaches for The Travelling Salesman Problem: A Survey from a Practical View

Bohan Zhang BOHAN.ZHANG@UWATERLOO.CA
The Cheriton School of Computer Science
University of Waterloo, Canada

Abstract

Reinforcement learning have recently showed promising results in solving NP-hard problems like Travelling Salesman Problem (TSP). However, people rarely discuss the practical advantage that RL has brought to us. In this paper, we survey recently published reinforcement learning (RL) approaches for solving the TSP. We compared the strength and limitation of selected approaches and conclude the most practical state-of-art RL method for TSP.

Keywords: Reinforcement Learning, Travelling Salesman Problem

1. Introduction

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" This problem is known as one of the most famous NP-hard (Papadimitriou, 1977) combinatorial and optimization (CO) problems - The Traveling Salesman problem. The formal definition of TSP is given below:

Definition *(2-Dimensional Euclidean Traveling Salesman Problem (TSP))* : Given a fully connected graph $G = (V, E)$, where $n = |V| > 1$ and each vertex $v_i, 1 \leq i \leq n$ has a weight 2-dimensional vector w_i, we are aiming to find a permutation of vertices $\pi = v^1, ..., v^n$ that minimizes the objective

$$L(\pi) = ||v^1 - v^n||_2 + \sum_{i=1}^{n-1} ||v^i - v^{i+1}||_2$$

where $|| \cdot ||$ denotes l2 norm.

TSP serves as the basis for many CO problems, including the Vehicle Routing Problem (VRP) and the TSP with Time Windows Problem (TSPTW). It is a critical topic in Operational Research(OR) because it has many real-world applications in logistics, genetics, planning, and scheduling (Lenstra and Kan, 1975). And it has not yet been resolved in polynomial time because of its NP-hardness. Numerous methods have been proposed by researchers over decades in an effort to tackle this problem as fast as possible. Lin-Kernighan heuristic (Lin and Kernighan, 1973) is one of the most efficient and effective algorithms which generates **near-optimal** solutions. And one of the most efficient **optimal** solution finders so far is Concorde TSP Solver, which is designed with complex hand-crafted branch

and cut heuristics and is way more time-consuming than a near-optimal solution finder.

As recent applications like DeepMind's AlphaGo and Facebook's Horizon (Gauci et al., 2018) have demonstrated outstanding results in completing challenging tasks using deep learning and reinforcement learning, OR researchers are actively using these techniques to address NP-hard problems. As one of the most famous NP-hard problems, TSP has received the most attention from the machine learning research community. In this paper, we introduce and compare some recent popular RL approaches and conclude the state-of-art methodology from a practical view. This work also complements an early survey (RL for CO problems) (Mazyavkina et al., 2021) but provides a more comprehensive analysis targeting only TSP. The audience of this work should have basic knowledge about reinforcement learning and deep learning.

2. Reinforcement Learning for TSP

In this section, we survey recent popular RL approaches to solve TSP. To systematically provide this survey, we will first introduce a unified neural combinatorial optimization pipeline similar to those introduced by (Joshi et al., 2022) and (Drori et al., 2020). The pipeline for applying deep reinforcement learning to CO problems consists of the following five elements:

- **Input Definition:** An instance of TSP is the input to the model. In particular, the model can take it as a fully connected graph, a processed sparse graph, or a simple ordered/unordered sequence of nodes.
- **Input Encoder:** A deep neural network that takes the input and maps each 2-dimensional node to a higher dimension. Depending on the architecture of the Input Encoder, it is possible to exploit the node neighbor information or global graph information when embedding the input.
- **Solution Decoder:** A deep neural network that takes the high-dimensional input embedding to generate a meaningful action/solution. Depending on the architecture of the Solution Decoder, the generated action/solution, for example, can be a complete tour or an improvement operation to an existing tour.
- **Learning Policy:** The policy that an RL agent learned and the RL algorithm that is used to guide the RL agent.
- **Inference Method:** After a model is trained, the Inference Method can be used to produce better results with the cost of more execution time. For example, we can run the trained stochastic model multiple times to generate different tours, and then we can pick the shortest one.

RL methods for tacking TSP can be roughly classified into two classes. One is the "*constructive*" approach, in which the RL agent generates a TSP solution in one shot by taking the TSP instance as input. The other is the "*improvement*" approach, in which an RL agent is trained to improve an initial TSP solution. Thus, running the model multiple times is essential for the "*improvement*" approach.

2.1 Summary

We will first present a summary of techniques. Some architecture details and MDP settings are omitted to avoid repeating introducing the same content and to highlight the contribution better. A full pipeline summary of the following 12 methodologies is listed in Table 1. As MDP modeling is usually not a key part of success, Table 3 containing a complete summary of MDP modeling is attached in Appendix A.

2.1.1 Constructive Methods

Parallelly, (Kool et al., 2018) and (Deudon et al., 2018) proposed a similar learning architecture. They took the input instance simply as a 2-dimensional vector data sequence and applied a transformer-based encoder (Veličković et al., 2017). The encoder first maps each input node i, $x_i \in R^2$, to a higher dimension and then feeds them into a multi-head attention (MHA) layer for information sharing between nodes. The input sequence size is not set to be fixed, and each node i ends up being mapped to a higher dimensional embedding, $h_i \in R^{128}$, after encoding. Their decoder is based on the attention mechanism. Iteratively, they feed all nodes embedding h_i whose corresponding node is not in the tour into an MHA layer along with a virtual context node and computes each city node's attention coefficient to the context node. The decoder will choose the node with the highest attention coefficient as the next node on tour. Therefore, such rollout decoding happens autoregressively but ensures that a solution is always valid. By adding stochasticity when decoding the next node, the RL agent benefits from exploration and can generate a different solution. The trained model will sample multiple different tours and then pick the best solution as the Inference Method. In the Markov Decision Process modeling of RL, the state is the TSP instance, and action is the TSP instance solution. After the decoder generates a full tour, a reward is given to the RL agent, which is the negative length of the complete solution tour. They used REINFORCE with a baseline algorithm taking randomly generated instances as training data to update the encoder-decoder network to maximize the expected reward.

Although the architectures are similar, (Deudon et al., 2018) applied the actor-critic method and trained an encoder-like critic network as the REINFORCE baseline during training, while (Kool et al., 2018) applied the t-test to determine the best model during training and achieved a higher expected reward by taking it as the baseline. For the context node during decoding, (Deudon et al., 2018) used partial tour's latest three nodes' embedding, which is missing the fact that the traveler needs to go back to the original start node. On the other hand, (Kool et al., 2018) used the latest node and the first node as context, which provides the network with more information. By comparison, (Kool et al., 2018)'s work outperforms the work of (Deudon et al., 2018). As an early work that achieves relatively good results, this attention model (AM) proposed by (Kool et al., 2018) has drawn the most attention from the community. Moreover, Therefore, AM is often used as the foundation of later works and as a performance baseline.

In AM, the model decodes the first visit city node based on a trainable START token. Like natural language processing(NLP), the first node heavily impacts the quality of the rest nodes. (Kwon et al., 2020) noticed this unreasonable design and pointed out that

any node should be equally good to be the first node. Therefore, on top of AM, they modified the decoding process of AM and proposed policy optimization with multiple optima (POMO) model. During decoding, the model randomly chooses a few (N) nodes, $n_1, ..., n_N$, as the first visit city node and then concurrently decodes different solution tours, $t_1 = (n_1...), ..., t_N = (n_N...)$, for one instance. Although the reward for each tour remains to be the negative total length and we will have $r(t_1), ..., r(t_N)$ for one instance, the REINFORCE baseline is changed to be the average of the tours $\frac{1}{N}[r(t_1) + ... + r(t_N)]$. This new baseline induces less invariance and makes the learning process more stable than AM. They also exploited the symmetric property of TSP, in which a solution should remain the same when rotating the TSP instance. For the inference method, they not only applied the sampling method but also introduced instance augmentation that rotates the 2-D instance to create new instances and then creates solutions. With the above new designs, POMO outperforms AM in the average length gap between optimal solutions and execution time for small TSP instances(<=100 nodes).

Although AM performs very well on small TSP instances when the training dataset is small-size instances, the training time becomes unacceptable when the training dataset is big instances(1000 nodes). A general way to avoid training is to directly apply AM trained from small to big-size instances. However, this approach performs even worse than the simplest heuristic algorithm (Nearest Neighbor). To address this, (Ma et al., 2019) replaced the transformer-based encoder in AM with a graph convolutional network (Kipf and Welling, 2016), which can better capture the information between nodes than the transformer. They treat the TSP instance as a fully connected graph, and the encoder will take it and create a global graph embedding. During decoding, they combined LSTM and attention decoder and changed the context node to be the two latest city nodes in the partial tour together with graph embedding. Also, they slightly changed the reward to the step-wise tour distance increment. Compared to giving the reward after a complete tour is fully constructed, this reward setting is less sparse and can stabilize the learning process. For small-size TSP instances, their model performs worse than AM. However, when their model is trained with small-size TSP instances and then tested on large TSP instances, it outperforms AM and achieves similar performance as the Nearest Neighbor algorithm.

Unlike the autoregressive decoding process in AM, (Emami and Ranka, 2018) trained an RL agent that generates a permutation solution in one shot. They define the input as a fixed $N \times 2$ matrix (N is the number of cities). The encoder is a fully connected layer with gated recurrent units, which outputs an $N \times N$ square matrix. The decoder is the Sinkhorn layer which consumes the square matrix and outputs a continuous doubly-stochastic matrix. A valid permutation matrix that corresponds to a TSP tour can be easily computed from the doubly-stochastic matrix. Different from all the above authors, they trained the RL agent using deep deterministic policy gradient (DDPG). Although their model solves TSP from a different perspective, it fails to outperform AM. Also, the trained model can only solve TSP with a fixed number of nodes, which makes it less useful in practice.

Unlike the search tree in Concorde or LKH that a neighbour is defined as a complete tour of another tour. Dynamic programming starts from one node and make decision of next

node based on a value-heuristic. (Cappart et al., 2021) pointed out that an RL agent can learn such heuristic functions in dynamic programming. Depending on search algorithms, value-based RL agents or policy-based RL agents can be trained to suit the algorithm. For *depth-first branch-and-bound search*, they train a DQN to learn the search heuristic and replace the heuristic computation directly with it. From the dynamic programming concept, it is intuitive that the state-action pair is the current tour-node pair, and the corresponding reward is the heuristic value. For *restart-based search*, in which the randomness serves as a critical component, they trained a policy network with proximal policy optimization(PPO) and let it satisfy the randomness requirements. Unlike the above works in which a generated partial solution will not be changed, their idea allows the search algorithm to discard the bad partial solution and leaves the final solution optimal, assuming the heuristic is well learned. Therefore, an inference method is hardly needed, which is more efficient than the above methods that require sampling. Although this method is novel, it suffers the same limitation as (Emami and Ranka, 2018) since the action space size needs to be fixed.

2.1.2 Improvement Methods

Like the heuristic algorithm, such as LKH, that starts from an initial solution, (Chen and Tian, 2019) lets the RL agent learn how to improve a solution. They simultaneously train two RL agents, the region picker, R_{Region}, and the rewrite rule picker R_{Rule}. R_{Region} is learning how to pick a set of nodes(e.g. two nodes) from all the nodes, and R_{Rule} is learning what operation needs to be performed on those selected nodes (perform a swap). They treat the TSP instance as a sequence of nodes and feed them into bidirectional LSTM, the encoder. Due to the nature of improvement, the MDP state set is defined as all possible solutions, and the reward is the length difference between two tours after an improvement. The action space of R_{Region} is a certain number of city nodes, and the action space for R_{Rule} is set to be a fixed number of rewriting rules. From the MDP formulation, we could realize that the trained model can generalize to all instances with different sizes. As the number of rewriting rules is fixed, they applied a simple fully connected layer as the decoder to output policy. They employed actor-critic to train their agents. Interestingly, R_{Region} is trained using deep Q-learning and simultaneously used as a critic network. As early work of improvement approaches, their model, the NeuRewriter, achieved similar good performance as AM while with a simpler model architecture. Also, this work can be very well applied to other CO problems, drawing great attention from the research community.

Although the NeuRewriter successfully mimics the traditional heuristic algorithm, it can easily get stuck in the local minimum since it always makes a similar level of change and hence lacks exploration in the inference step. To address this, (Lu et al., 2019) added a perturbation operator in the decoder to increase the search space. The decoder in their model is offered a global view of the training procedure to detect if the current improvement loop is stuck at the local minimum. Besides outputting improvement policy, the decoder can destroy and re-permutate the current solution so that the training process and inference are given much more exploration than the NeuRewriter. They changed the state definition to provide the network with more information by feeding additional past H actions and corresponding rewards. As constructive models output a solution in one shot, training mul-

tiple constructive models is nearly the same as sampling multiple solutions by one model. However, for the improvement method, they empirically proved that following the best decision from multiple models can achieve better performance when inferencing. By changing the value of H, they can train multiple models and use the ensemble effect in the inference step. Their model outperforms the NeuRewriter and AM with the ensemble technique and perturbation operator.

When the model takes the input as a sequence, the relative node position is essential in learning. All the above works that define the input as a sequence did not fully exploit such information. Inspired by NLP, (Wu et al., 2019) first computed the node-wise sum of the sinusoidal positional encodings (PE) of the sequence and the nodes' embeddings and then fed them into a transformer-based AM encoder to create the final node embedding. The embedding is then forwarded to the compatibility matrix layer decoder to create a square $N \times N$ matrix output. The matrix entry index (i, j) means that the agent will perform 2-opt on node pair (i, j). And the output action will be the matrix entry with the highest value. Because of the complexity of the model, the execution time is way worse than AM. However, this model outperforms AM and the NewRewriter.

(Ma et al., 2021) pointed out that instead of fusing node embedding and PE, pushing the RL agent to learn from embedding and PE separately will introduce less noise and avoid incompatible correlations. Also, the sinusoidal PE cannot carry the cyclic property of TSP. Thus, they proposed to use the novel cyclic positional encoding(CPE) based on cyclic Gray codes instead of sinusoidal PE and proposed Dual-Aspect Collaborative Transformer(DACT) model. DACT roughly consists of two models of (Wu et al., 2019). The first model consumes node embedding and outputs the node-pair square matrix. The second model is used to consume CPE and output another square matrix. Two models will share some information during encoding and decoding. Once all proposals from two aspects are collected, they are fed into an attention layer to create the final matrix. They trained the model with PPO, a novel algorithm more stable than REINFORCE. Since they have exploited the cyclic property and separated the position learning, their model outperforms POMO and (Wu et al., 2019), becoming the neural method with the minimum optimal gap.

To fix the limitation of (Wu et al., 2019) that it can only learn 2-opt due to the square matrix structure, (d O Costa et al., 2020) proposed a model to learn general k-opt. While the above improvement works take input as sequence, they take the current tour solution as a graph and apply GCN to encode it to create the graph context embedding, which can better capture the edge information, and hence PE is not necessarily needed. Then, they applied an AM-like decoder to generate a sequence of N nodes. Different from in AM, where N equals the number of nodes of the TSP instance, N is set to be the number of nodes that a k-opt operation should be performed on. For comparison with previous work, they set N to be 2 representing a 2-opt operation. Their model outperformed (Wu et al., 2019) in the optimal gap and also achieved a lower execution time than AM.

All the above methods require neural network structure, so a training procedure is necessary. (Zheng et al., 2021) proposed Variable Strategy Reinforced LKH(VSR-LKH) that

Table 1: Summary of Techniques

Paper	Encoder	Decoder	Inference	Learning Policy
(Kool et al., 2018)	Transformer	Attention	Sampling	Solution Rollout REINFORCE with baseline
(Deudon et al., 2018)	Transformer	Attention	Sampling + 2-opt	Solution Rollout Actor-Critic
(Kwon et al., 2020)	Transformer	Attention	Sampling + Augmentation	POMO Solution Rollout REINFORCE with baseline
(Ma et al., 2019)	GCN	Attention + LSTM	Sampling + 2-opt	Rollout REINFORCE with baseline
(Emami and Ranka, 2018)	MLP	SinkHorn	2-opt	Solution Permutation Actor-Critic
(Cappart et al., 2021)	GAT	MLP	NA	DP Heuristic DQN/PPO
(Chen and Tian, 2019)	Bidirectional LSTM	MLP	Multi-Run	Region, Rule Actor-Critic
(Lu et al., 2019)	Transformer	Attention + Perturbation	Multi-Run + Ensemble	Rule REINFORCE with baseline
(Wu et al., 2019)	Transformer	Compatibility	Multi-Run	2-opt Matrix Actor-Critic
(Ma et al., 2021)	Dual-Transformer	Dual-Compatibility	Multi-Run	2-opt Matrix Actor-Critic
(d O Costa et al., 2020)	GCN	Attention	Multi-Run	k-opt Rollout Actor-Critic
(Zheng et al., 2021)	NA	NA	NA	LKH-α-value Monte Carlo + 1-step TD

combines RL and LKH to solve TSP without a neural network. LKH starts with a random initial solution and performs 2-opt to improve it iteratively. For 2-opt, MDP modeling suits very well. The state can be a city node, the action set is all other cities, and the transition is defined by LKH, which is the next city to perform 2-opt. During iteration, LKH changes the cost matrix of TSP by adding a penalty term to the distance of two cities to maximize the lower bound of the optimal solution. The difference between the penalty before 2-opt and after 2-opt is given to the RL agent as a reward. For one city node, LKH will compute α-values for other cities to represent how likely they should be connected. This value is designed to be learned by the RL agent as Q-value. Therefore, the Q-table will be a 2-D square matrix and is initialized meaningfully based on LKH. They employed the Monte Carlo method and one-step temporal difference to update the Q-table, including off-policy Q-learning and on-policy Sarsa. Compared to all the above methods, their method can work well on instances of any size without pretraining. This RL-LKH combined algorithm outperforms LKH with a similar running time.

Note that in CO problems, how to handle constraints is major study when applying RL. (Ma et al., 2019) and (Cappart et al., 2021) have proposed good architecture to deal TSPTW and CVRP with constraints. Due to the space limitation, we cannot present their details.

2.2 Analysis

Considering traditional methods, Concorde is slow but guarantees an optimal solution, and LKH is fast but only finds a near-optimal solution. What practical advantage has RL brought to us compared to these algorithms? For small-size TSP, POMO is the best-

Table 2: Performance Comparison

Model	TSP-20		TSP-50		TSP-100	
	Gap	Time	Gap	Time	Gap	Time
Concorde	0.00%	3m	0.00%	10m	0.00%	1h
LKH	0.00%	38s	0.00%	5m	0.00%	20m
POMO	0.00%	3s	0.03%	16s	0.15%	1m
DACT	0.00%	3m	0.00%	10m	0.09%	29m

performed model among selected constructive methods, and DACT is the best-performed model among selected neural improvement methods. Both methods are successful due to exploiting the symmetric property of TSP, so it shows that a qualified and comprehensive context passed to the RL agent should be the most critical element of the neural network approaches. Interestingly, none of the above 12 methods is successful due to simply applying another learning method. Therefore, studying the impact of different RL learning algorithms is still an open problem. Table 2 presents a performance comparison between POMO, DACT LKH, and Concorde, solving 10000 small instances. For time-critical tasks such as a highly dynamic environment, the constructive method is more useful in practice since it can output a solution within a second, and the inference technique allows it to do its best within any required time limit. Neural improvement methods outperform constructive methods and can be more efficiently trained. However, it requires much more time to generate satisfying solutions. Compared to LKH, neural improvement methods do not show an advantage in practice. Also, all neural approaches share two same practical problems. The first one is that they are hard to scale. As the size of the problem gets bigger, the performance worsens (Joshi et al., 2022). Another problem is that the training time increases exponentially as the instance size increases. In contrast, as a perfect replacement for LKH, VSR-LKH scales well replying on traditional search logic. It also does not require pre-training and expert domain knowledge, making it a state-of-art RL application on TSP in practice.

3. Conclusion

In this survey, we learned about recent popular RL learning architectures tackling TSP, along with their strength and limitation. We concluded that combining traditional heuristic algorithms and RL is most useful in practice compared to purely deep neural network based learning architecture. We also concluded the advantage of constructive methods in tackling time-critical TSP problems. Based on the above analysis, the following future research directions are suggested:

- Study the impact of applying different RL learning algorithms.
- Improve constructive methods to tackle large-size TSP.
- Constructive methods may help reduce the execution time of improvement-based algorithms by generating a better initial solution than random initialization.

Table 3: Summary of MDP

Paper	State	Action	Reward
(Kool et al., 2018)	Instance Sequence	Full Tour	Tour Length
(Deudon et al., 2018)	Instance Sequence	Full Tour	Tour Length
(Kwon et al., 2020)	Instance Sequence	Multi Full Tour	Average Length
(Ma et al., 2019)	Partial Tour	City Node	Increment Length
(Emami and Ranka, 2018)	Instance Sequence	Full Tour	Tour Length
(Cappart et al., 2021)	Instance Sequence + Partial Tour	City Node	Algorithm Defined Cost
(Chen and Tian, 2019)	Full Tour	Improvement Rule	Tour Length Difference (Current Tour, State Tour)
(Lu et al., 2019)	Instance Sequence + Full Tour + History Action-Reward	Improvement Rule	+1 if Improved
(Wu et al., 2019)	Full Tour	Node Pair	Tour Length Difference (Current Tour, Best Tour)
(Ma et al., 2021)	Full Tour + Position	Node Pair	Tour Length Difference (Current Tour, Best Tour)
(d O Costa et al., 2020)	Full Tour + Best Tour	Node Pair	Tour Length Difference (Current Tour, Best Tour)
(Zheng et al., 2021)	City Node	City Node	LKH Defined Cost Difference

References

Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687, 2021.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

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.

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.

Iddo Drori, Anant Kharkar, William R Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 19–24. IEEE, 2020.

Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. *arXiv preprint arXiv:1805.07010*, 2018.

Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, Xiaohui Ye, Zhengxing Chen, and Scott Fujimoto. Horizon: Facebook's open source applied reinforcement learning platform. *arXiv preprint arXiv:1811.00260*, 2018.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, pages 1–29, 2022.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016. URL `https://arxiv.org/abs/1609.02907`.

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

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Jan Karel Lenstra and AHG Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

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

Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2021.105400. URL `https://www.sciencedirect.com/science/article/pii/S0305054821001660`.

Christos H. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(77)90012-3. URL `https://www.sciencedirect.com/science/article/pii/0304397577900123`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL `https://arxiv.org/abs/1710.10903`.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems, 2019. URL https://arxiv.org/abs/1912.05784.

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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12445–12452, 2021.