Study on the Solution of Traveller Salesman Problem with Deep Reinforcement Learning

Site Bai, Qilei Zhang
{Bai123, Zhan3599}@purdue.edu.

Abstract

In this project, we apply Deep Reinforcement Learning to solve the Traveler Salesman Problem and two TSP variants with constraints. Firstly, we identified S2V-DQN as the state-of-the-art Deep RL algorithms that combines Graph Neural Networks to solve the TSP, re-implemented it in PyTorch and achieved identical results reported in the original paper. Secondly, we apply S2V-DQN to solve the TSP with Time Windows and achieved comparable results with other operations research tools. Thirdly, we test the performance of S2V-DQN on the TSP with Pickup and Delivery, and the results are unsatisfying. We analyze the reason may be that the brute-force masking scheme on the output layer is not enough for complex constraints, which may need elaborate design to be intrinsically embedded into the deep networks.

1 Introduction

Traveler Salesman Problem (TSP) is one of the most classic combinatorial optimization problems and are studied in multiple fields. Since its first formulation, many solutions have been proposed [Osaba et al., 2020]. The solving algorithms can be generally classified into three approaches. The first one is the exact approach, which searches for the entire solution space but is currently known to take beyond polynomial time [Karp, 1972, IBM, 2016]. The second approach is the heuristic approach that applies various approximation algorithms to reach a near-optimal solution [Aarts et al., 2003]. The third approach is the learning approach, which applied machine learning to solve the TSP. Recent advances in Deep Reinforcement Learning (RL) have stood out among the learning approaches. In this project, we conduct an in-depth investigation into how Deep Reinforcement Learning is applied to solve the TSP, and two variants of the TSP with additional constraints: TSP with Time Windows (TSPTW) and TSP with Pickup and Delivery (TSPPD).

1.1 Literature Review

Utilizing Deep RL to solve the TSP has been an active field [Bello et al., 2017, Dai et al., 2017, Deudon et al., 2018, Kool et al., 2019, Ma et al., 2020, da Costa et al., 2020]. Bello et al., 2017 built their model from the Pointer Networks proposed by Vinyals et al., 2015, in which a supervised learning recurrent neural network (RNN) architecture was introduced to solve the TSP, regarding it as a sequence-to-sequence transformation. Bello et al., 2017 adopted the Pointer Networks as the parametric presentation of their Deep RL policy, and trained it following a similar way to the advantage actor-critic (A2C) algorithm [Mnih et al., 2016]. Kool et al., 2019 and Deudon et al., 2018 extended the idea by embedding an attention mechanism, and solved the model with the parametric version of the REINFORCE algorithm [Williams, 1992]. It has been the state-of-the-art on-policy RL algorithm for Vehicle Routing Problem. For off-policy Deep RL algorithms, Dai et al., 2017, Ma et al., 2020, da Costa et al., 2020 have introduced graph neural networks (GNN) to encode the TSP as a structure-to-vector transformation and achieved better results. However, Bello et al., 2017, Dai et al., 2017 and da Costa et al., 2020 only tested their performances on vanilla TSP, while real-world problems may impose various constraints. In this project, we identify the S2V-DQN [Dai et al., 2017] as the state-of-the-art off-policy approach that introduced GNN to the TSP, and apply it to solve the TSP with Time Windows and TSP with Pickup and Delivery.

1.2 Broad Impact

The TSP problem is a common and practical problem that has a wide range of applications. For example, it enables the logistics distribution organization to get the best driving route in the fastest time, and to achieve the purpose of fast distribution in this manner. However, in practical applications, there will be various unique nuances and constraints [Marques et al., 2019], and an increasing amount of data sets are also obstacles to obtaining the best solution. Over the years, due to the great success of deep learning in many fields, letting machines learn autonomously how to solve problems is quite promising. It not only saves money and time, but may also generate a more feasible solution than manual design [Cook, 2011].

2 Problem Formulation of the TSP with Constraints

2.1 Vanilla TSP

Given N nodes $\{v_1, v_2, \dots, v_N\} \subset \mathbb{R}^2$, the TSP is to find the optimal route that covers the minimal distance to visit each coordinate exactly once [Lawler et al., 1986]. We formalize the TSP as the following optimization problem [Ma et al., 2020]:

$$\min_{\sigma} L(\sigma, V) = \sum_{i=1}^{N} ||v_{\sigma(i)} - v_{\sigma(i+1)}||_2 \tag{1}$$

$V = \{v_1, v_2, ..., v_N\}$. σ represents the permutations of the coordinates. $\sigma(1) = \sigma(N+1)$, $\sigma(i) \in 1, 2, ..., N$, $\sigma(i) \neq \sigma(j)$ for any $i \neq j$.

2.2 TSP with Time Windows

Baker presented the first mathematical model for *symmetric* TSP with Time Windows (TSPTW) where total tour duration is minimized [Baker, 1983]. Here, *symmetric* means for each edge has the same costs in both direction. Baker's model stands as the only formulation for simply minimizing completion time [Kara and Derya, 2015]. The model appears as follows:

$$\begin{aligned} \min\ & t_{n+1} - t_0 \\ \text{s. t. } & t_i - t_0 \geq t_{0i}, i \in (1, 2, \ldots, n) \\ & |t_i - t_j| \geq t_{ij}, i \in (2, 3, \ldots, n), 1 \leq j < i \\ & t_{n+1} - t_i \geq t_{i0}, i \in (1, 2, \ldots, n) \\ & t_i \geq a_i, i \in (1, 2, \ldots, n) \\ & t_i \leq b_i, i \in (1, 2, \ldots, n) \\ & t_i \geq 0, i \in (1, 2, \ldots, n+1) \end{aligned} \tag{2}$$

where t_{ij} is the travel time from node v_i to node v_j. a_i is the earliest visit (service begins) time of the node v_i. b_i is The latest visit time of the node v_i. $[a_i, b_i]$ is time window of the node v_i.

2.3 TSP with Pickup and Delivery

In general, there are four constraints in the *symmetric* TSP with Pickup and Delivery Problem. First constraint is there is an connection between origin and destination nodes $\{+0, -0\}$ with a cost of $c_{+0,-0} = 0$. It is a convention so that every feasible route can form a Hamiltonian circle. The second constraint require that each node is entered and exited in all feasible routes, but by itself leaves open the possibility of disconnected subtours [O'Neil and Hoffman, 2018]. The third constraint set the subtour elimination requirement. The last constraint requires that every goods should be picked up before dropped off, which is a primary difference compared to the regular TSP problem [Ruland and Rodin, 1997]. In summary the formulation is as follows:

$$\begin{aligned} \min\ & \sum_{(i,j)\in E} c_{ij} x_{ij} \\ \text{s. t. } & x_{+0,-0} = 1 \\ & x(\delta(i)) = 2, \forall i \in V \\ & x(\delta(S)) \geq 2, \forall S \subset V \\ & x(\delta(S)) \geq 4, \forall S \subset V, \{+0, -i\} \subset S, \{-0, +i\} \subset V \setminus S \end{aligned} \tag{3}$$

where V is the union of V_+ and V_- with the addition of origin and destination nodes $\{+0, -0\}$. $x_{ij} \in \{0, 1\}$ is a binary decision variable for each $e(i, j) \in E$ with the value $x_{ij} = 1$ if the edge $e(i, j)$ is in a solution and 0 otherwise. $\delta(S) = \{e(i, j) \in E | i \in S, j \notin S\}$ is the cutset containing edges that connect $S \subset V$ and $\overline{S} \subset V$.

3 METHOD

3.1 REINFORCEMENT LEARNING FORMULATION FOR TSP

Reinforcement Learning operates through interacting with the environments and receiving rewards. It is formalized based on a Markov Decision Process which can be defined by tuple $(\mathcal{S}, \mathcal{A}, \pi, r, \gamma)$. $\mathcal{S}$ represents the state space and $\mathcal{A}$ denotes the action space. $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$ defines the policy that the agent follows. Reward function $r : \mathcal{S} \rightarrow \mathbb{R}$ is the reward from the environment and $\gamma \in (0, 1)$ is a discount factor. In the context of the vanilla TSP, the elements are represented in the following settings:

- State is defined as the set of previously visited nodes: $s_i \in \mathcal{S}$, $s_i = \{v_{\sigma(i)}\}_{k=1}^{i}$.
- Action is defined as the possible selections of the next node: $a_i \in \mathcal{A}$, $a_i = v_{\sigma(i+1)}$.
- Reward is defined as the negative cost of traveling from one node to another: $r_i = -||v_{\sigma(i)} - v_{\sigma(i-1)}||_2$.

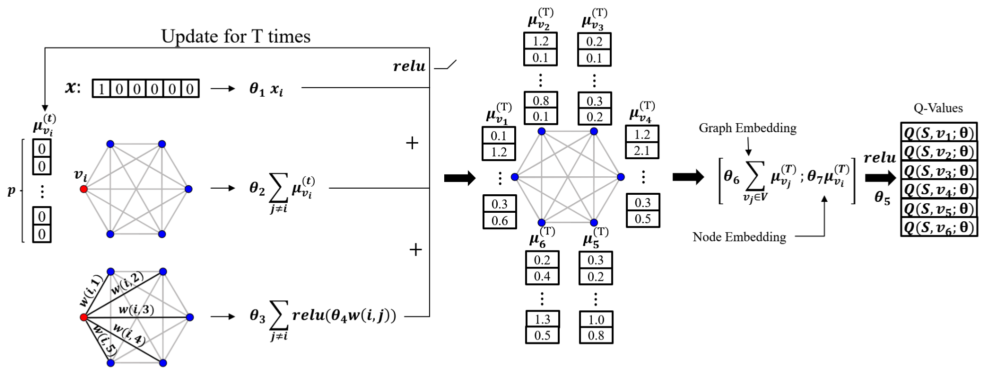

Figure 1: Model of S2V-DQN

3.2 S2V-DQN

S2V-DQN [Dai et al., 2017] proposes to embed the graphical input of the TSP to a vector representation of each node with the help of the Structure2Vec [Dai et al., 2016] network. It then train the network using the node vector embedding as input with the classic Deep RL algorithm DQN [Mnih et al., 2015].

3.2.1 Structure2Vec

Given a graph $G = (V, E)$, the set of nodes V can be divided in to S and $\overline{S}$, which represent the set of visited nodes and the set of those not yet visited. The Structure2Vec network initializes a p-dimensional vector representation $\mu_{v_i}^{(0)}$ for each node v_i. The embedding is updated leverageing three parts of information. The first is a state vector that records weather a node has been visited. The second part is the sum of the node vectors of the adjacent nodes. The third is the $relu$ activated sum of the weights on the connected edges of the node. The node vectors are updated for T times by the relu activated sum of these three parts as follows:

$$\mu_{v_i}^{(t+1)} \leftarrow relu\left(\theta_1 x_i + \theta_2 \sum_{j \neq i} \mu_{v_j}^{(t)} + \theta_3 \sum_{j \neq i} relu\left(\theta_4 w(i,j)\right)\right), \tag{4}$$

in which x_i is a binary variable s.t. $x_i = 1$ for $V_i \in S$ and $w(i, j)$ is the weight or cost over the edge $e(i, j)$. Then we get a graph with each node as a vector, embedding the graphical information. The node vectors are summed up as the embedding of the entire graph. The graph embedding is concatenated with each node embedding, to compute the final output of the Q-value for each node:

$$Q\left(S, v_i; \Theta\right) = \theta_5^\top relu\left(\left[\theta_6 \sum_{v_j \in V} \mu_{v_j}^{(T)}; \theta_7 \mu_{v_i}^{(T)}\right]\right), \tag{5}$$

in which $[\cdot;\cdot]$ represents concatenation, $\Theta = \{\theta_i\}_{i=1}^{7}$ and T is a hyper-parameter controlling how many updates to learn the vector representation.

3.2.2 The S2V-DQN Algorithm

DQN [Mnih et al., 2015] is a widely applied Deep Reinforcement Learning algorithm. It parameterizes the Q function in Q-learning with a deep neural network and learns a mapping between the input states and the Q-values of different actions. For S2V-DQN, the Q-function is parameterized as the S2V network introduced in Section 3.2.1. The ϵ-greedy mechanism is also applied, that is, to choose the optimal action with the maximum Q-value under probability $(1 - \epsilon)$ and take actions randomly under probability ϵ. It is a strategy that's commonly applied with DQN to encourage exploration. The parameters of the Q-function is optimized by minimizing TD-error as the loss . The complete learning process of S2V-DQN is shown in Algorithm 1.

3.2.3 Adaptation to TSPTW and TSPPD

Training Process for the TSPTW: For TSPTW, the training process basically follows that in the vanilla TSP, with a few modifications regarding the time window constraints. As introduced in Section 2, the visit time t_i of each node has to fall in the time window $[a_i, b_i]$. If the agent chooses to visit node v_i and arrives at a time prior to a_i, the agent

Algorithm 1: S2V-DQN with ϵ Greedy

```
Result: Optimized $\Theta$
Initialize Experience Replay Buffer
for episode = 1 to $L$ do
    Generate graph $G$;
    Initialize the state $S_1 = ()$;
    for step $t = 1$ to $T$ do
        $v_t = \begin{cases} random\ v_i \in \overline{S_t} & w.\ p.\ \epsilon \\ \arg\max_{v_i \in \overline{S_t}} Q\left(S_t, v_i; \Theta\right) & otherwise. \end{cases}$
        $S_{t+1} := (S_t, v_t)$
        Add $(S_t, v_t, R_t, S_{t+1})$ to replay buffer;
        Sample random data batch from replay buffer;
        Update $\Theta$ with SGD by optimizing
          $\left(\gamma \max_{v'} Q\left(S_{t+1}, v'; \Theta\right) + r\left(S_t, v_t\right) - Q\left(S_{t+1}, v_t; \Theta\right)\right)^2$
    end
end
```

has to wait for a period of $a_i - t_i$ to enter and then leave node v_i. Such waiting time penalty is added to the reward for the TSPTW.

$$r_i = -||v_{\sigma(i)} - v_{\sigma(i-1)}||_2 + \min\{t_i - a_i, 0\} \tag{6}$$

And we regulate that a node v_i cannot be visited when the time has already passed b_i. So it's still quite possible that the agent will fail to traverse all the nodes. We'll introduce a training data generation mechanism that assigns reasonable time windows to ensure there is definitely a feasible solution for the agent to learn. During the training process, we would still mask out the nodes that violate the time window constraint and directly discard such training instances.

S2V-DQN for the TSPPD: The key feature of the TSPPD is that the nodes come in pairs with precedence relationships. That means the nodes for pickup and the nodes for delivery come in pairs and the agent has to pickup before delivery. So in the training for S2V-DQN, we pair the nodes by their pickup and delivery relationships, and at the beginning, the nodes for delivery are masked out, leaving only the pickup ones optional, until the corresponding node for pickup has been visited. The masking scheme is illustrated in Figure 2.

4 Experiment

We firstly re-implement the S2V-DQN model in PyTorch and test it on vanilla TSP to verify its effectiveness and the correctness of our code, then apply S2V-DQN to solve the TSPTW and the TSPPD, which are problems not considered in the original papers. We firstly introduce how the training data is generated in order to abide by the con-

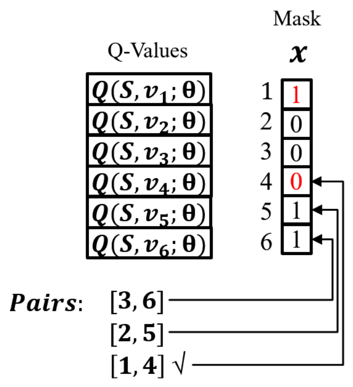

Figure 2: Masking Scheme of S2V-DQN for TSPPD.

straints, then explain the choice of baselines, and finally demonstrate the experimental result and analysis.

4.1 Training Data Generation

For the vanilla TSP, we generate fully connected graphs with a graphical Python package called NetworkX [Hagberg et al., 2008] containing specified number of nodes, and generate their coordinates from randomly from $[0, 1]^2$. The weights of the edges are then computed by the euclidean distance between the node coordinates.

For the TSPTW, apparently we need a time window $[a_i, b_i]$ for each node v_i in the graph, such that $t_i \in [a_i, b_i]$. If the time windows are generated randomly, it's possible that there may not even be a feasible solution, i.e. we cannot traverse all nodes within their time windows. The strategy here is to firstly ensure a feasible solution and generate time windows w.r.t. the constraints. Based on the graph generated as those in the vanilla TSP, we solve the vanilla TSP using a heuristic algorithm called 2-opt local search [Aarts et al., 2003]. Then following the route, we compute the time of arrival for each node based on previous nodes and sample a time window in the following way: $a_i \sim \max\{0, \mathrm{U}(t_i - 2, t_i)\}$ and $b_i \sim \mathrm{U}(t_i, t_i + 2)$. In the training process, we discard the actions that would violate the constraints of time windows, and if a solution cannot be found, it's regarded as a training failure.

For the TSPPD, we randomly pair the nodes in the generated graph and build a dictionary to record their corresponding pickup and delivery relationship. When choosing the next node to visit, only the pickup node and the delivery node with corresponding pickup node visited are optional.

4.2 Baselines

We choose two baselines for this task. First, intelligent optimization algorithms are widely applied to solve NP-hard problems, so we pick a classic one, the Ant Colony

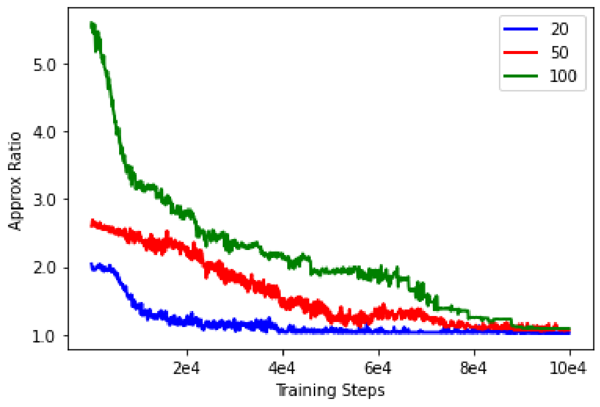

Figure 3: Convergence Curve of the Approximation Ratio of S2V-DQN on Vanilla TSP

Table 1: S2V-DQN Test Result of Vanilla TSP of Different Sizes

Number of Nodes	Approx Ratio (Ours)	Approx Ratio (Original)	Test Running Time
20	2.01%	1.47% (15-20)	0.11s
50	6.13%	5.33% (40-50)	1.78s
100	8.84%	7.01% (50-100)	4.53s

Optimization [Colorni et al., 1992], as our baseline. Another useful tool called the OR-tools [Google, 2016] recently developed by Google is popular in Operations Research, so it is picked as our second baseline. For the ground truth solution, we apply CPLEX, a state-of-the-art exact linear programming approach for solving combinatorial optimization problems.

4.3 Result

4.3.1 Re-implementation of S2V-DQN on the Vanilla TSP

We firstly compare our PyTorch re-implementation of S2V-DQN to the C++ Cuda implementation result reported in the original paper on the vanilla TSP to verify the effectiveness of S2V-DQN and the correctness of our implementation. It is tested on graphs with 20, 50 and 100 nodes respectively. We evaluate the result by the travel distance approximation ratio of the S2V-DQN solution to the ground truth solution. We generate a separate validation dataset of size 100 for parameter tuning and a test set of size 100 to compute the average approximation ratio. The convergence curve is shown in Figure 3 and the final results are shown in Table 1.

From the result, we see that the approximation ratio and the running time increase with the size of the graph, but the performance is basically identical to that reported in the original paper. There is a slight gap between our result and the original percentage because, as shown in the table, the results in the original paper are averaged among different graph sizes, for example, 15 to 20 nodes, which will be lower than the result for 20 nodes. Thus, in general, S2V-DQN is effective in solving the vanilla TSP and our implementation in PyTorch is correct.

4.3.2 Result for the TSPTW

We observe the travel cost, i.e. the length or travel time of the solutions given by different algorithms for the TSPTW with 20 nodes and 50 nodes. The result is computed by the average of 20 graphs in the TSPTW Benchmark Dataset [López-Ibáñez and Blum, 2010]. Given the CPLEX solution as the ground truth, the result for S2V-DQN, ACO and OR-Tools is demonstrated in Figure 4. From the result, we can see

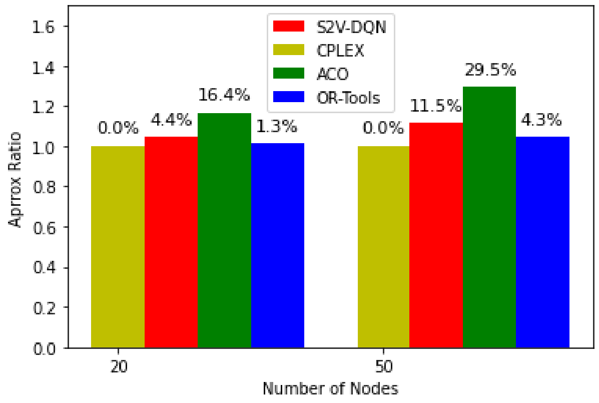

Figure 4: Approximation Ratio on TSPTW 20 and TSPTW 50 for CPLEX, S2V-DQN, ACO and OR-Tools.

that even though S2V-DQN is less optimal than the OR-Tools result, it's reasonable to say that S2V-DQN can learn a descent policy even after transferring from Vanilla TSP to TSPTW, and it yields an obvious improvement comparing to traditional intelligent algorithms like ACO. Thus, we can say that S2V-DQN is well applicable to the TSP with some simple constraints like TSPTW.

4.3.3 Result for the TSPPD

Grubhub sample instances have been collected as the test set for the TSPPD. This dataset includes the problem instances of observed meal delivery requests and actual travel time estimates, which are considered as pre-computed edge costs [Grubhub, 2018]. The result of applying S2V-DQN to the TSPPD is shown in Table 2. The approximation ratio of S2V-DQN is many times larger than that of OR-Tools, showing that S2V-DQN has failed to learn a comparable policy to exact or heuristic solvers.

Our analysis is that we are only imposing masking scheme on the output layer of the Deep Q-Network, which may not be enough for the agent to capture the pairing and the precedence relations between nodes. A more desirable way would be embed such pairing and precedence relations into the network architecture as a Deep RL model that is specifically designed for the TSPPD task. In fact, as we are carrying our project, progress has been made by other works in this regard. Li et al., 2021 have proposed a Deep RL model based on an on-policy algorithm with the help of Attention mechanisms. But nevertheless, S2V-DQN is of wider range applications and can be applied to multiple combinatorial problems.

Table 2: S2V-DQN Test Result for TSPPD with 20 Nodes

Number of Nodes	Approx Ratio
CPLEX	0%
S2V-DQN	61.88%
OR-Tools	3.52%

4.4 Hyper-parameters

We document the hyper-parameters of the S2V-DQN model for different tasks in Table 3

Table 3: Hyper Parameters for S2V-DQN

Task	Learning Rate	γ	ϵ	T	p	Batch Size	Buffer Size
Vanilla TSP 20	0.01	0.97	0.2	3	64	64	10000
Vanilla TSP 50	0.005	0.99	0.5	4	64	128	10000
Vanilla TSP 100	0.005	0.99	0.5	4	64	128	10000
TSPTW 20	0.001	0.97	0.5	4	64	64	10000
TSPTW 50	0.001	0.99	0.5	4	64	128	10000
TSPPD 20	0.001	0.97	0.2	4	64	64	10000

5 Conclusion and Future Work

In this project, we studied how to apply Deep Reinforcement Learning to solve the Traveler Salesman Problem. Specifically, we identified S2V-DQN as the state-of-the-art Deep RL algorithms that combines Graph Neural Networks to solve the problem, re-implemented it in PyTorch and achieved identical results reported in the original paper. Moreover, we applied S2V-DQN to solve the TSP with simple constraints like TSP with Time Windows and found it effective regarding problem of this kind. We also explored the performance of S2V-DQN on the TSP with more complex constraints like the TSP with Pickup and Delivery, and reported unsatisfying results. We analyzed that the cause may be imposing the brute-force masking scheme on the output layer while the problem itself may need elaborate design of the deep networks that can intrinsically embed the constraints. This points out the future direction of this work and many other on-going works are also good references [Li et al., 2021].

6 CONTRIBUTIONS

Site Bai implemented the code, conducted the experiments, and wrote major parts of the final report. Qilei Zhang worked on the problem formulation of the TSPTW and TSPPD, studied the broad impact of this work, collected the datasets, prepared slides and presented the work. Works like discussion of the topics, literature review and study of the method are completed together.

REFERENCES

[Aarts et al., 2003] Aarts, E., Aarts, E., and Lenstra, J. (2003). *Local Search in Combinatorial Optimization*. Princeton University Press.

[Baker, 1983] Baker, E. K. (1983). An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5):938–945.

[Bello et al., 2017] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net.

[Colorni et al., 1992] Colorni, A., Dorigo, M., Maniezzo, V., Varela, F., and Bourgine, P. (1992). Distributed optimization by ant colonies.

[Cook, 2011] Cook, W. J. (2011). *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press.

[da Costa et al., 2020] da Costa, P. R. d. O., Rhuggenaath, J., Zhang, Y., and Akcay, A. (2020). Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In Pan, S. J. and Sugiyama, M., editors, *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 of *Proceedings of Machine Learning Research*, pages 465–480, Bangkok, Thailand. PMLR.

[Dai et al., 2016] Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711. PMLR.

[Dai et al., 2017] Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6351–6361, Red Hook, NY, USA. Curran Associates Inc.

[Deudon et al., 2018] Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. (2018). 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.

[Google, 2016] Google (2016). Or-tools. `https://developers.google.com/optimization/routing`.

[Grubhub, 2018] Grubhub (2018). Tsppd test instance library. `https://github.com/grubhub/tsppdlib`.

[Hagberg et al., 2008] Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

[IBM, 2016] IBM (2016). Cplex, ibm ilog cplex optimization tools. `https://www.ibm.com/analytics/cplex-optimizer`.

[Kara and Derya, 2015] Kara, I. and Derya, T. (2015). Formulations for minimizing tour duration of the traveling salesman problem with time windows. *Procedia Economics and Finance*, 26:1026–1034. 4th World Conference on Business, Economics and Management (WCBEM-2015).

[Karp, 1972] Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40:85–103.

[Kool et al., 2019] Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

[Lawler et al., 1986] Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1986). The traveling salesman problem: A guided tour of combinatorial optimization. *Journal of the Operational Research Society*, 37(5):535–536.

[Li et al., 2021] Li, J., Xin, L., Cao, Z., Lim, A., Song, W., and Zhang, J. (2021). Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–10.

[López-Ibáñez and Blum, 2010] López-Ibáñez, M. and Blum, C. (2010). Beam-aco for the travelling salesman problem with time windows. *Computers Operations Research*, 37(9):1570–1583.

[Ma et al., 2020] Ma, Q., Ge, S., He, D., Thaker, D., and Drori, I. (2020). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. In *AAAI Workshop on Deep Learning on Graphs: Methodologies and Applications*.

[Marques et al., 2019] Marques, R., Russo, L., and Roma, N. (2019). Flying tourist problem: Flight time and cost minimization in complex routes. *Expert Systems with Applications*, 130:172 – 187.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[Osaba et al., 2020] Osaba, E., Yang, X.-S., and Del Ser, J. (2020). Chapter 9 - traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics. In Yang, X.-S., editor, *Nature-Inspired Computation and Swarm Intelligence*, pages 135 – 164. Academic Press.

[O'Neil and Hoffman, 2018] O'Neil, R. J. and Hoffman, K. (2018). Exact methods for solving traveling salesman problems with pickup and delivery in real time. *Optimization-Online. org: http://www. optimization-online. org/DB_HTML/2017/12/6370. html. Accessed*, 9.

[Ruland and Rodin, 1997] Ruland, K. and Rodin, E. (1997). The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & mathematics with applications*, 33(12):1–13.

[Vinyals et al., 2015] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 2692–2700, Cambridge, MA, USA. MIT Press.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.