POINTER NETWORKS WITH Q-LEARNING FOR COMBINATORIAL OPTIMIZATION

Alessandro Barro
Politecnico di Milano
Milan, MI 20133
alessandro1.barro@mail.polimi.it

May 15, 2024

ABSTRACT

We introduce the Pointer Q-Network (PQN), a hybrid neural architecture that integrates model-free Q-value policy approximation with Pointer Networks (Ptr-Nets) to enhance the optimality of attention-based sequence generation, focusing on long-term outcomes. This integration proves particularly effective in solving combinatorial optimization (CO) tasks, especially the Travelling Salesman Problem (TSP), which is the focus of our study. We address this challenge by defining a Markov Decision Process (MDP) compatible with PQN, which involves iterative graph embedding, encoding and decoding by an LSTM-based recurrent neural network. This process generates a context vector and computes raw attention scores, which are dynamically adjusted by Q-values calculated for all available state-action pairs before applying softmax. The resulting attention vector is utilized as an action distribution, with actions selected hinged to exploration-exploitation dynamic adaptibility of PQN. Our empirical results demonstrate the efficacy of this approach, also testing the model in unstable environments.

1 INTRODUCTION

Combinatorial Optimization (CO) tasks are inherently complex and often unfold in dynamic environments. Traditional attention-based methods like Pointer Networks (Ptr-Nets) [7] excel in handling sequence-based problems by focusing on locally optimal decisions, yet they may overlook long-term consequences.

To address these limitations, we integrate Ptr-Nets with Q-learning, a model-free method that enhances decision-making without depending on accurate environmental models. This contrasts with Ptr-Nets combined with model-based reinforcement learning (RL) methods [6] [8], which require known environmental dynamics. Our approach is particularly advantageous in unpredictable environments where future states are unknown or variable.

In this paper, we introduce the Pointer Q-Network (PQN), which combines the model-free policy approximation capabilities of Q-learning with the sequential decision-making strengths of Ptr-Nets. PQN aims to improve decision-making in CO tasks by effectively balancing local optimality with strategic long-term planning. The experimental section demonstrate PQN's ability to manage complex, dynamic optimization challenges more effectively than conventional methods.

2 BACKGROUND AND PROBLEM FORMULATION

2.1 Travelling Salesman Problem (TSP)

The symmetric TSP is a benchmark for studying complex decision-making under constraints and is ideal for evaluating the performance of the PQN against other methods. Following the DFJ formulation, let n cities $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ and $\mathbf{X} = \{\mathbf{x}_{ij}\}_{i,j=1}^{n}$ denote edges. If $\mathbf{c}_{ij} > 0$ represents the cost linked to each edge $\mathbf{x}_{ij}$, the TSP is defined by the

following objective

$$\min_{i,j} \mathcal{J}(\mathbf{x}_{ij}, \mathbf{c}_{ij}) = \min_{i,j} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \mathbf{x}_{ij}\mathbf{c}_{ij} \tag{1}$$

subject to

$$\sum_{i=1, i\neq j}^{n} \mathbf{x}_{ij} = 1, \quad \forall j = 1, \ldots, n, \tag{2}$$

$$\sum_{j=1, j\neq i}^{n} \mathbf{x}_{ij} = 1, \quad \forall i = 1, \ldots, n, \tag{3}$$

$$\sum_{i\in S} \sum_{j\in S, j\neq i} \mathbf{x}_{ij} \leq |S| - 1, \quad \forall S \subset \{1, \ldots, n\}, |S| > 2 \tag{4}$$

These constraints ensure flow conservation and eliminate subtours, maintaining the integrity of the tour.

2.2 MDP Graph Embedding

To effectively apply the PQN to CO problems such as the TSP [5], we embed the corresponding graph within a MDP synergistically merging sequence-to-sequence modeling with critic-only reinforcement learning (RL). The MDP framework is represented as the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

The state space $\mathcal{S}$ comprises subsets of visited cities at each timestep $t = 0, \ldots, T$, with the initial state $s_0 = \{\mathbf{v}_{\text{start}}\}$, where $\mathbf{v}_{\text{start}}$ is the starting city.

The action space $\mathcal{A}(s_t)$, denoting the set of feasible cities that can be visited next from the current state s_t, is defined as $\mathcal{A}(s_t) = \{\mathbf{v} \in \mathbf{V} \setminus s_t : (i, \mathbf{v}) \in \mathbf{E} \text{ for some } i \in s_t\}$.

The transition probability $\mathcal{P}$, which is deterministic in this context, specifies that $\mathcal{P}(s_{t+1}|s_t, a_t) = 1$ if $s_{t+1} = s_t \cup \{a_t\}$, indicating that the next state s_{t+1} is obtained by adding the city $a_t \in \mathcal{A}(s_t)$ to the set of already visited cities s_t.

The reward function $\mathcal{R}(s_t, a_t)$ is defined to penalize the travel cost $\mathcal{R}(s_t, a_t) = -c_{ia_t}$, where $i \in s_t$ and $(i, a_t) \in \mathbf{E}$, thus motivating the PQN to minimize the travel distance. We simplify this notation later in the article as $\mathcal{R}(s_t, a_t) = r_t$.

The goal is to learn an optimal policy $\pi^*(a_t|s_t)$, which determines the best action distribution from s_t and influences the decision-making process of the hybrid model. We will explore this aspect further in subsequent sections.

3 PROPOSED APPROACH

3.1 Pointing Mechanism

The pointing mechanism dynamically determines the most promising next city to visit by leveraging the learned attention scores [11] and handling variable-sized inputs. Within this mechanism, the current state s_t is decoded by an LSTM layer [10] into $h_t \in \mathbb{R}^k$, a higher-dimensional representation that also serves as the hidden state of the LSTM, where k is the dimensionality of the hidden layer.

Given the action space $\mathcal{A}(s_t)$, which contains all feasible next cities at time step t, the network computes an attention score for each potential action $a \in \mathcal{A}(s_t)$. The attention score u_{ta} for each action a_t is calculated using the formula

$$u_{ta} = v^T \tanh(W_1 h_t + W_2 e_a), \tag{5}$$

where

- e_a is the LSTM-embedded vector corresponding to the city represented by action a
- W_1 and W_2 are trainable weight matrices that project h_t and e_a into a joint feature space
- v is a trainable vector that transforms the output of the activation function into the final attention score

The computed logits u_{ta}, or 'raw pointers', are then utilized within the Q-learning framework to establish the action distribution, guiding the selection of the next city to be visited.

3.2 Q-Learning

In our model, rather than directly applying a softmax to the raw attention scores u_{ta}, we first compute Q-values for each state-action pair (s_t, a_t) using a dedicated Q-Network, parameterized by θ_Q. The Q-values are derived using the Bellman equation [3], which evaluates immediate rewards and the discounted value of future states:

$$Q(s_t, a_t; \theta_Q) = r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a'; \theta_Q) \tag{6}$$

where $\gamma \in [0, 1]$ is the discount factor that balances immediate and future rewards. The notation a' refers to potential future actions from s_{t+1}.

At each iteration $t = 0, \ldots, T$, the Q-values are iteratively refined using the update rule:

$$Q(s_t, a_t; \theta_Q) \leftarrow Q(s_t, a_t; \theta_Q) + \eta \left[r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a'; \theta_Q) - Q(s_t, a_t; \theta_Q) \right] \tag{7}$$

where η is the learning rate. This update rule enables the Q-values to increasingly reflect more accurate estimations of expected future rewards as training progresses.

Instead of Q-Learning, we opt for neural network Q-value approximation [1], employing a FNN directly on the context vector $c_t = \tanh(W_1 h_t + W_2 e_a)$. Significant techniques adopted to stabilize training and expedite convergence time of the Q-Networks include

- *Target Networks* help stabilize training and address fluctuations stemming from highly correlated data, as typically encountered in path-solving problems. The target network maintains the structure of the original network and is parameterized with θ_Q^-, which are updated less frequently than θ_Q. $Q(s_t, a; \theta_Q^-)$ substitutes $Q(s_{t+1}, a'; \theta_Q)$ in the temporal difference calculation.
- *Experience Replay* [12] involves storing transitions $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in a replay buffer and sampling a batch of transitions to update the Q-Network. This technique breaks the correlation between consecutive learning updates and allows for more effective learning from diverse parts of the environment.

3.3 PQN

To integrate these Q-values into the pointing mechanism, we combine them with the logits u_{ta} to obtain the final attention scores via softmax.

$$\tilde{\pi}(a_t | s_t) = \frac{\exp(u_{ta} \Phi(Q_{ta}, t))}{\sum_{i=1}^{k} \exp(u_{ti} \Phi(Q_{ti}, t))} \tag{8}$$

The *crystal* function Φ, designed to modulate the influence of Q-values over time $t \in [0, T] \subset \mathbb{N}$, is crucial for adjusting the decision-making process during training dynamically. It directly acts on the distribution's entropy, due to the direct relationship with logits.

$$\Phi(Q_{ta}, t) = Q_{ta} \varphi(t) \tag{9}$$

Where $\varphi \in C^\omega$ is an analytical function scaling Q_{ta} over time. Since this one gets relevant while training evolves, φ should be a time transformation s.t. $\frac{\Delta \Phi}{\Delta t}(Q_{ta}, t) \geq 0$. Overall, reasonable Φ choices include time-linear $\Phi_{\text{lin}}(Q_{ta}, t) = Q_{ta}(\alpha t + \beta))$ with $\alpha, \beta \geq 0$, and time-exponential $\Phi_{\text{exp}}(Q_{ta}, t) = Q_{ta} \exp(\frac{\gamma}{T - t + 1})$ with $\gamma > 0$.

Traditional Q-learning aims to derive the optimal policy π^* from refined Q-values, focusing purely on long-term reward maximization. In contrast, our model interprets the attention scores $\tilde{\pi}$ as a soft policy. This dual-purpose formulation allows the scores to not only focus the LSTM-generated attention but also act as a probabilistic policy that guides decision-making. The attention scores integrate immediate contextual insights with learned value estimations, effectively balancing exploration and exploitation in a dynamic environment.

An interesting observation is that even although $\tilde{\pi}$ is not retrieved conventionally but refined alongside with Q-values, PQN still maintains model-free outlines, not requiring simulations or any knowledge about how actions affect the environment beyond what is observed through interactions.

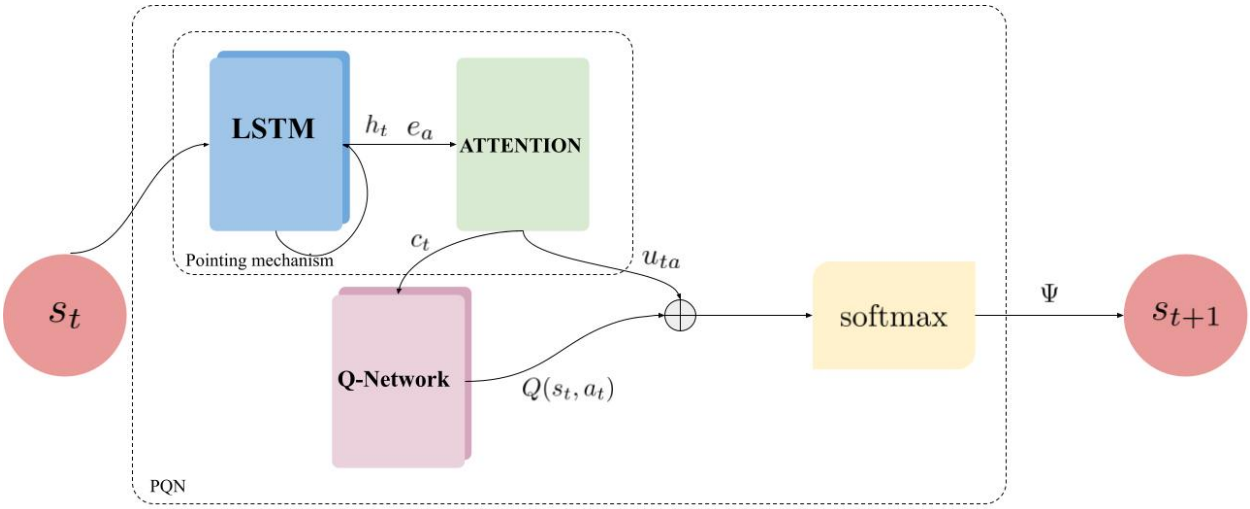

Figure 1: PQN's sequence flow chart

Algorithm 1 PQN

Initialize network parameters θ, θ_Q randomly
Initialize target network parameters $\theta_Q^- \leftarrow \theta_Q$
Initialize replay buffer $\mathcal{D}$
for each episode **do**
    Initialize the sequence state s_0
    for $t = 0$ to T **do**
        Encode state s_t using LSTM to get h_t
        For each action $a \in \mathcal{A}(s_t)$, compute:
            $u_{ta} = v^T \tanh(W_1 h_t + W_2 e_a)$
        Compute Q-values $Q(s_t, a_t; \theta_Q)$ for all $a_t \in \mathcal{A}(s_t)$
        Compute policy $\tilde{\pi}(a_t|s_t)$ using softmax:
            $\tilde{\pi}(a_t|s_t) = \frac{\exp(u_{ta}\Phi(Q_{ta},t)}{\sum_{i=1}^{k}\exp(u_{ti}\Phi(Q_{ti},t))}$
        Select action a_t based on policy $\tilde{\pi}(a_t|s_t)$
        Execute action a_t, observe reward r_t and new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
        Sample random minibatch of transitions from $\mathcal{D}$
        Update θ_Q by minimizing the loss:
            $L(\theta_Q) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})}\left[\left(r + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_Q^-) - Q(s, a; \theta_Q)\right)^2\right]$
        Every C steps reset $\theta_Q^- \leftarrow \theta_Q$
    end for
    Update θ, θ_Q parameters of LSTM, Attention, Q-Network
end for

4 EXPERIMENT

4.1 Implementation Details and Setup

The experiments were conducted on limited resources. We used a 2022 MacBook Air with an Apple M2 chip and 8 GB of RAM, running macOS Ventura 13.4. The experimental framework was implemented in Python 3.11.4, utilizing key libraries including NumPy for numerical operations, TensorFlow for neural network operations, NetworkX for graph-related functionalities, and Matplotlib for data visualization.

The study focused on the symmetrical euclidean TSP, specifically the TSP20 and TSP50 instances, representing tours of 20 and 50 cities respectively. For each one, we defined three different problem instances, for training, evaluation and testing respectively.

The Pointer Network (Ptr-Net) employed a single-layer LSTM with 128 or 256 hidden units, trained using TF's ADAM v2.11 optimizer on Lin-Kernighan Heuristic (LKH) [9] generated target sequence, with a batch size of 64 and a learning rate of 0.1. The network weights were initially set using a uniform distribution. For Q-value approximation, a feedforward neural network with one hidden layer was used, configured similarly to the main Ptr-Net model.

4.2 Comparison Metrics

We utilize a set of metrics that collectively evaluate both the efficiency and effectiveness of the model. The selected metrics include

- *Average Cumulative Cost* $\mathcal{J}(\mathbf{x}_{ij}, \mathbf{c}_{ij}) = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \mathbf{x}_{ij}\mathbf{c}_{ij}$.
- *Final Output Sequence* τ, refers to the sequence permutation of cities visited in the TSP as determined by the PQN at the end of an episode.
- *Accuracy*: Measured by the percentage deviation σ of the tour lengths generated by the PQN from those produced by the established LKH method, calculated as $\sigma_{\text{B}} = \frac{|\tau_{\text{PQN}}|-|\tau_{\text{B}}|}{|\tau_{\text{B}}|}$, where $|\tau_{\text{B}}|$ is the LKH benchmark tour length.

4.3 Empirical Results

For initial experiments, we examined TSP20 5. Training was conducted on 5 randomly generated TSP20 instances, each with a sequence length $|\tau| = 20$, across $|\mathcal{E}| = 30$ epochs and $T = 100$ time steps per epoch . Graphs are fully connected. In the beginning, a linear crystal function $\Phi_{\text{lin}}(Q_{ta}, t) = Q_{ta}t$ was used.

The performance of the PQN was compared against a standard Ptr-Net trained in parallel with PQN but utilizing distinct gradients from the Q-Network during training as well as different logits during training and evaluation. The same hyperparameters for the sequence model were shared between PQN and Ptr-Net. As mentioned, the output sequences were benchmarked using the LKH heuristic.

	PQN	Ptr-Net	LKH
$\mathcal{J}$	4.0870	4.2110	3.8562
σ_{B}	94.0137%	90.8028%	N/A

Table 1: Comparison of results for TSP20

Additionally, the Chebyshev norm of the crystal function $||\Phi_{\text{lin}}||_{\infty}$ 2 was monitored, displaying a comb-shaped pattern indicative of the model's balancing between exploration and exploitation.

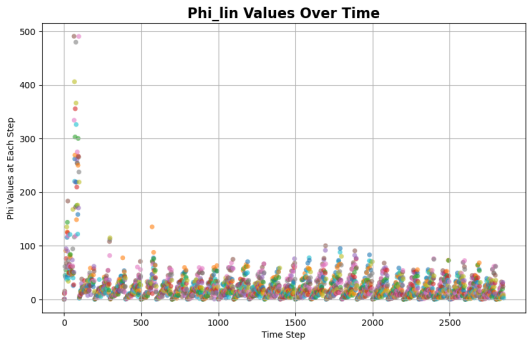

Figure 2: $||\Phi_{\text{lin}}||_{\infty}$ over time steps t

We observe that, as expected, decision-making increasingly relied on Q-values at later iterations. Furthermore, exploratory tests with $\gamma = 2$ time-exponential crystal function Φ_{exp} showed denser and higher contribution from Q-values correlating with periods of loss fluctuations, suggesting a self-stabilization effect in the training process 3.

Essentially, in complex decision-making scenarios depicted by high loss, Q-values demonstrate enhanced sensitivity in capturing environmental dynamics. This phenomenon highlights PQN's ability to not only learn but also dynamically adapt its strategy based on environmental interplay.

To test out this theory, we intentionally introduced a perturbation δ_m [13] on the train TSP instances costs $\mathbf{c}_{ij_m} + \delta_m$ for $m = 1, \ldots, s$ (where s is the number of samples), in the epochs range $[5, 10]$. We randomly generated perturbations from a uniform distribution $\delta_1, \ldots, \delta_s \sim \mathcal{U}(\alpha_u, \beta_u)$ with $\alpha_u = 0.9$ and $\beta_u = 1.1$, which correspond to 10% variation on distances. Training on 20-nodes perturbated TSP instance was conducted, and led to temporary instabilities as suggested from the loss profiles in $[500, 1000]$ time range, which exactly correspond to the selected epochs range.

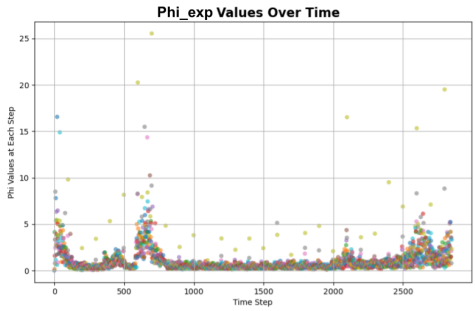

Figure 3: $||\Phi_{\text{exp}}||_\infty$ over time steps t

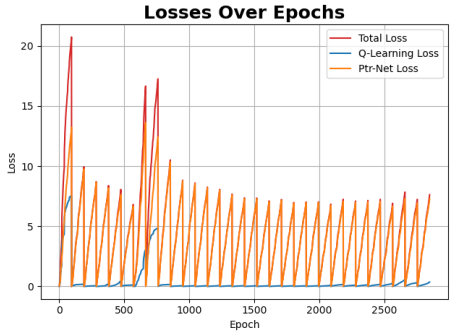

Figure 4: Loss profile under Φ_{exp}

While both models show they are being sensible to perturbations, $||\Phi_{\text{exp}}||_\infty$ spikes suggest an increment of PQN's deterministic behaviour in correspondence of environment fluctuations, exhibiting gradual adaptation and conducting the model's permutations towards optima. The evaluation of the models on new non-modified TSP20 instances display PQN's outstanding self-stabilization and adaptation capabilities.

	PQN	Ptr-Net	LKH
$\mathcal{J}$	4.5913	6.7803	4.1360
σ_{B}	88.9897%	36.0643%	N/A

Table 2: Comparison of results for perturbated TSP20

On the other hand, TSP50 5 is instanced by 12 samples TSP50 instances, each with a sequence length $|\tau| = 50$. There are $|\mathcal{E}| = 90$ epochs and $T = 500$ time steps per epoch. This TSP version produced the following outcomes

	PQN	Ptr-Net	LKH
$\mathcal{J}$	7.9563	8.7114	6.0735
σ_{B}	68.9997%	56.5680%	N/A

Table 3: Performance comparison for TSP50

While maintaining a lower cumulated cost by 0.7551, PQN conserved a closer proximity to the optimal solution, significantly higher than Ptr-Net by 12.4317 percentage points. This highlights PQN's enhanced learning capabilities and strategic depth in more vast and complex scenarios.

As a computational note, the time complexity of PQN for the current problem setup can be described as $\mathbf{O}(n^2)$. In particular both Ptr-Net and Q-learning share the need compute through every vertex pair, resulting in n^2 operations each.

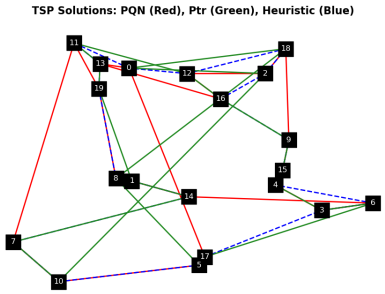

Figure 5: Path visualization for TSP20

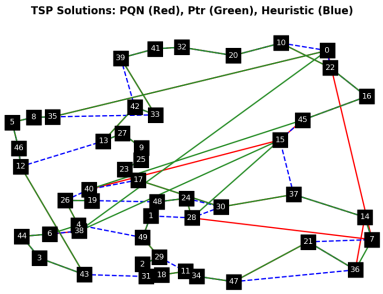

Figure 6: Path visualization for TSP50

5 CONCLUSION

In this study, we introduced the Pointer Q-Network (PQN), a neural architecture that integrates model-free Q-learning with Ptr-Nets to enhance decision-making in combinatorial optimization tasks. By utilizing the crystal function Φ, similarly to the softmax temperature, PQN uniquely blends Q-values with attention mechanisms, addressing both immediate accuracy and strategic long-term planning in challenges such as the TSP.

Our empirical results demonstrate that PQN significantly improves solution quality and computational efficiency, especially in intricate CO landscapes characterized by less-predictable and mutable future states.

PQN's exploration of larger TSP instances remains limited, primarily due to computational constraints, underscoring a critical area for future development. Further research could also explore adapting the crystal function Φ to more complex temporal dynamics or dependencies, potentially improving PQN's strategic depth.

We also look forward to exploring further applications of PQN in problems that differ from the TSP, including other CO contexts and beyond.

6 ACKNOWLEDGEMENTS

We express sincere gratitude to Professor Cristiana Bolchini and Park Kwanyoung who provided exceptional insights for the paper's structure; Valerio Romano Cadura offered valuable guidance and ideas; Professor Anand Avati for his quality teachings in machine learning.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning, Nature (2015).

[2] Sutton, R. S., Barto, A. G. et al.: Reinforcement learning: An introduction (2018).

[3] Watkins, C. J., Dayan, P. et al.: Q-learning (1992). Machine learning.

[4] A. Holmberg, W. Hansson: Combinatorial Optimization with Pointer Networks and Reinforcement Learning (2021)

[5] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, Le Song: Learning Combinatorial Optimization Algorithms over Graphs (2017).

[6] T. D. Barrett, W. R. Clements, J. N. Foerster, A. I. Lvovsky.: Exploratory Combinatorial Optimization with Reinforcement Learning (2019).

[7] O. Vinyals, M. Fortunato, N. Jaitly: Pointer Networks (2015).

[8] Q. Ma Suwen, Ge Danyang, He Darshan, T. I. Drori: Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning (2019).

[9] K. Helsgaun: An effective implementation of the Lin–Kernighan traveling salesman heuristic, European Journal of Operational Research (2000)

[10] S. Hochreiter, J.Schmidhuber: Long short-term memory, Neural computation (1997)

[11] A. Graves, G. Wayne, I. Danihelka: Neural turing machines (2014)

[12] Revisiting Fundamentals of Experience Replay W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, W. Dabney: Revisiting Fundamentals of Experience Replay (2020)

[13] J. P. Cohoon, J. E. Karro, W. N. Martin, W. D. Niebel: Perturbation Method For Probabilistic Search For The Traveling Salesperson Problem (1998)