AlphaRouter: Bridging the Gap Between Reinforcement Learning and Optimization for Vehicle Routing with Monte Carlo Tree Searches

Won-Jun Kim
Department of Future Mobility
Hanyang University
Seoul, Replic of Korea
glistering96@hanyang.ac.kr

Kichun Lee*
Department of Industrial Engineering
Hanyang University
Seoul, Replic of Korea
skylee@hanyang.ac.kr

Abstract

Deep reinforcement learning (DRL) as routing problem solvers have shown promising results in recent studies. However, a gap between computationally-driven DRL and optimization-based heuristics exists. While a DRL algorithm for a certain problem is able to solve several similar-kind problem instances, traditional optimization algorithms focus on optimizing one specific problem instance. In this paper, we propose an approach, AlphaRouter, which bridges the gap between reinforcement learning and optimization for solving routing problems. Fitting to routing problems, our approach first proposes attention-enabled policy and value networks consisting of a policy network that produces a probability distribution over all possible nodes and a value network that produces the expected distance from any given state. We modify a Monte Carlo tree search (MCTS) for the routing problems, selectively combining it with the routing problems. Our experiments demonstrate that the combined approach is promising in yielding better solutions compared to original RL approaches without MCTS and good performance comparable to classical heuristics.

*corresponding author

Preprint. Under review.

1 Introduction

In NP-hard combinatorial optimization (CO) problems, finding global optimum solutions is computationally infeasible. Instead of finding global optima, numerous heuristics showed promising results. Despite the high effectiveness of heuristics, their application in real-life industries is often hindered by a variety of problems and uncertain information. Indeed, the heuristics, mathematically originated, are dependent on problem formulations for their proper application. However, an exact formulation of constraints is quite challenging in reality as some constraints are rapidly changing and highly stochastic even in a distributional sense. For instance, a few constraints vanish at a time and other constraints newly enter with coefficients that, being unknown, should be estimated by an assumed distribution and a certain procedure. As real-life domains are entangled with various participants and requirements, some constraints are too complex to formulate. In such situations, thus, particularly when simulation is possible as in a game, the approach of RL has recently attracted attention in the literature and industry.

Mostly, the heuristics aim to solve one specific problem. That is to say, heuristics made for capacitated vehicle routing problems (CVRP), for example, cannot apply to bin-packing problems. To deal with versatile constraints and complex problems, the use of deep neural network architectures coupled with RL, called as DRL, has recently been considered effective [19, 20]. The DRL approach is flexible as translating a problem into a reinforcement learning framework is straightforward without necessiating mathematical solutions but by appropriately defining both state and reward and running computational simulations. In the long run, the ultimate goal of the DRL approach is to find a new and computational way to solve a complex problem that surpasses the performance of mathematically exact algorithms and their heuristics [11].

Nonetheless, the current stage of DRL's performance is not statisfactory enough to reach the performance of heuristic solvers, and is still on the journey to the ultimate goal. Our work is to improve DRL performance by attempting to reduce the gap between heuristic solvers and DRL. Motivated by the AlphaGo series [26, 29, 30], we propose a deep-layered network for RL equipped with selective application of a Monte Carlo Tree Search (MCTS), a general framework applicable to various types of CO problems. We modify some components of MCTS for the application to routing domains that are different from a game. Unlike the AlphaGo series, we found that a mere application of the MCTS to every action choice is inefficient, thereby proposing a selective application of our MCTS for routing problems. In this paper, the domain is routing problems that will be extensible to other problems.

Indeed, the neural network architectures have been introduced before in RL contexts [9, 17, 19]. However, to the best knowledge of ours, no attempt has been made to improve network architectures for routing problems integrated with customized MCTS. The introduced RL framework is quite beneficial if the resulting network is applicable to other problem types by using the same network architecture as suggested in [19, 20, 38]. The contributions of our paper are threefold: (1) We propose a deep-layered neural network architecture, fitting to the routing problem, with a policy gradient

using a value network; (2) We propose a new MCTS strategy, demonstrating that the integration and selective application of our MCTS into the neural network architecture improves the solution quality; (3) We also demonstrate the effect of activation to improve the solution quality. In short, the main focus of this paper is to propose an effective RL architecture with a modified MCTS strategy for routing problems and to improve search performance. Notably, although we have proposed a neural network architecture specialized to routing problems, any neural network architecture containing a policy and a value, which will be described in later sections, can be integrated into MCTS [27].

We organize the rest of the paper as follows: In Section 2, we provide an overview of previous works related to combinatorial optimization, routing, and MCTS. In Section 3, we briefly introduce a general formulation of capacitated VRP and our problem's objective. In Section 4, we expound our proposed approach, AlphaRouter. In Section 5, we present our experimental results.

2 Related works

Routing problems are among the most well-known set of problems in combinatorial optimization. A traveler salesman problem (TSP), one of the simplest routing problems, is to find the sequence of nodes with the shortest distance. On the other hand, a vehicle routing problem (VRP), similar to TSP, is a routing problem with the concept of depots and demand nodes. Numerous variants of VRP exist in the literature such as VRP with time windows and VRP with pickup and delivery among many [18]. In this paper, we focus on the capacitated VRP (CVRP), where a vehicle has a limit on its loading amount. Although some variants handle multiple vehicles, we only consider one vehicle for simplicity.

Traditionally, solutions for these problems mainly belong to two types: using math-based approaches like mixed integer programming or applying carefully designed heuristics for a specific type of problem. An example of the latter is the Lin-Kernighan heuristic [21]. In fact, through the past decade, hybrid genetic searches with advanced diversity controls were introduced and applied to various CVRP variants successfully, greatly improving the computation time and performance[36, 37]. We agree that the current stage of DRL is hard to reach nor pass the performance of the analytically driven heuristics, but still, there should be efforts to solve the problem using less mathematics-entangled methods such as dynamic programming and stochastic optimization.

Recent research on neural networks for routing problems can be broadly categorized into two approaches based on the type of input they use: graph modules and sequential modules. Graph modules take a graph's adjacency matrix as an input and employ graph neural networks to solve routing problems, which are naturally suited to graph structures of routing problems [9, 20]. In contrast, sequential models use a list of node coordinates as input and are designed to be compatible with certain types of exact solution inputs. In this paper, we focus on the sequential module approach.

The pointer network [38] is the early model for solving routing problems. It suggested a supervised way to train a modified Seq2Seq (sequence-to-sequence) network [32] with an early attention

mechanism [23] by producing an output that is a pointer to the input token. However, a significant disadvantage of the pointer network is that one cannot obtain enough true labels to train large problems since routing problems are NP-hard. To overcome this limitation, the approach in [3] introduced the RL method for training neural networks using the famous and simple policy gradient theorem [33].

Similar to machine translation evolving from Seq2Seq to Transformers [35], routing has also adopted the Transformer architecture in the research [17] using both the attention layer to encode the relationships between nodes and the encoding in the decoder to produce a probability distribution for most promising nodes. Replacing the internal neural network only, they kept the training of RL the same as in [3].

In addition to designing neural architectures, some works focus on the search process itself. For example, the work [19] introduced a parallel in-training search process, named POMO, based on attention network designs. POMO algorithm assigns a different start node for several rollouts and executes the multiple episodes concurrently. An episode or rollout can be understood as a process in which a vehicle travels to the next customer until all customers are visited. Among the many episodes, they selected one best solution as the final solution. Although it does not introduce any more parameters to the model, the size of the input is bound to be $O\left(N^2\right)$ usually, where N is the number of total nodes in the problem. Another approach is to adjust the weights of a pre-trained model during inference to fit the model into a single problem instance, as proposed in [14].

MCTS is a decision-making algorithm commonly used in games, such as chess, Go, and poker. The algorithm selects the next actions by simulating the game and updates the decision policy using data from the simulation. The original algorithm consists of four phases: selection, expansion, rollout, and backpropagation. In the selection phase, the algorithm starts at the root node and recursively goes down to a child node, maximizing upper confidence bound (UCB) scores. This score balances exploration and exploitation in the selection process by considering the visit counts and the average value (i.e., average winning rate) gained on that node. A more detailed explanation of UCB is in reference to [5].

When the selection phase ends and the current node is a leaf node, the expansion phase is executed, in which a new child node is appended to the leaf node following a specific policy named “expansion policy". Then, the rollout phase, using “rollout policy", simulates the game until it ends and gathers the result (i.e., win or lose). In the backpropagation phase, by backtracking the path (sequence of selected nodes), the evaluated result from the rollout policy is updated for each node. For example, the updates are increasing visit counts by one and updating the average win rate with the result from the rollout phase. We note that there are two different policies used in the original MCTS, but in our MCTS implementation, only the expansion policy exists and the rollout policy is replaced by the value network. We describe more on this in Section 4.

Since MCTS shares similarities with routing problems in that sequential decision-making involves, we have adopted MCTS as an additional search strategy for routing problems. In the game of Go, the next move is selected sequentially based on the board situation. In routing, the next node to visit is

selected sequentially based on the current position and other external information, such as nodes' location and demands. Thus, the similarity between Go and routing is obvious.

Neural networks have been successfully integrated with MCTS in AlphaGo [29], and AlphaGo Zero achieved even better results by introducing a self-play methodology for training the network [30]. The first AlphaGo used two different neural networks for "expansion" and "rollout" inducing a computational burden because of many recursive calls for the "rollout" network until the end of the game. This problem is solved by AlphaGo Zero, in which one call for the value network predicts the expected result from any given state (or node). This was originally gathered in the rollout phase. In short, only one call for the value network replaced numerous calls for the rollout network, saving substantial computations. Our work adopts this idea for efficient MCTS simulation.

Some more recent works that integrated neural network with MCTS not only exists in the area of games but also in a Q-bit routing challenge [31]. In the final stage of writing this paper, we have found a recent paper that utilizes a similar approach to our method but for a slightly different problem [22]. Using a neural network with MCTS is therefore a fine approach to solve a problem.

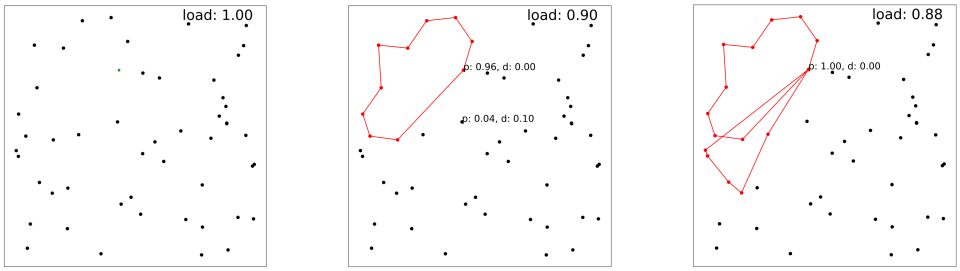
3 Preliminaries

Before expounding our work, we introduce a formulation of CVRP with one vehicle to connect with our routing problem. We notice that TSP can be easily formulated from CVRP by modifying some conditions.

We start with a set of n customers, and each customer i, $i = 1, \cdots, n$, has a predefined positive demand quantity q_i. To fulfill the customers' demands, the vehicle starts its route at the depot node, indexed as 0. The vehicle must visit each customer only once, and its capacity cannot be more than Q^{max}. In conventional settings, assumably , Q^{max} is set to be sufficiently large to fulfill all customers' demand. However, in practice, a vehicle may start with small Q^{max} due to lack of information and its load should be refilled. We observe that the problem formulation itself below cannot reflect the vehicle refilling, but we aim to handle this situation, as an example of dynamic routing problem [15], using our RL approach in the next section.

For example, as described in Figure 1, we present a scenario where the vehicle refills with a smaller amount of load for a reason after its first subroute is created and solved with DRL. In the figure, dots represent customer nodes and red dots denote the nodes visited which are also connected by the red line. Figure 1a represents the initial setting with $Q^{max} = 1$, Figure 1b shows the situation after the vehicle and is refilled with a slightly different $Q^{max} = 0.9$, and Figure 1c shows the routing result with $Q^{max} = 0.9$. Usually, to handle these dynamics of the environment, a complicated mathematical formulation or expert engineering techniques are needed [15]. However, with DRL, one can just adjust Q^{max}, which needs only one line to be changed in our implementation.

As a graph representation of the problem is common in the literature, we also represent the problem using a graph $G(V, E)$ where $V = \{0, 1, \cdots, n, n+1\}$, meaning all the nodes in the problem: 0



(a) initial (b) first refill (c) second refill

Figure 1: Routing result in dynamic load settings

and $n+1$ are the *same* depot node. The last node, $n+1$, is just an extra term for the ease of the formulation as the final depot of a tour. We define π_t to be the node visited at time $t, t \geq 0$ with $\pi_0 = 0$ and a tour, $\boldsymbol{\pi}_{t_1,t_2}$, from time t_1 up to t_2 is defined as a sequence of visited nodes: for example, $\boldsymbol{\pi}_{0,T} = [\pi_0 = 0, \pi_1 = 2, \pi_2 = 7, ..., \pi_T = n+1]$, in which T is the last time point in the tour. The terms route and tour are used interchangeably. Additionally, $E = \{ (i,j) \mid i,j \in V \}$ means all the edges from all node combinations. Note that the demand of depot node q_0 is 0, meaning $q_0 = q_{n+1} = 0$. We also introduce a binary decision variable x_{ij} which is 1 if there is a direct route from customer i to j, and 0 otherwise. The distance of edge (i,j) is denoted by c_{ij}. The cost $C(\boldsymbol{\pi}_{0,t})$ is the cumulative distance calculated so far at t, given the sequence of visited nodes: $C(\boldsymbol{\pi}_{0,t}) = c_{\pi_0,\pi_1} + \cdots c_{\pi_{t-1},\pi_t}$, and $C(\boldsymbol{\pi}_{0,0}) = 0$. Lastly, on the route up to a visit of node $j \in V$, a continuous variable y_j represents accumulated demands and is dependent on the decision variable x_{ij}: for instance, on tour $\boldsymbol{\pi}_{0,3} = [\pi_0, \pi_1, \pi_2, \pi_3]$, $y_{\pi_3} = q_{\pi_0} + q_{\pi_1} + q_{\pi_2} + q_{\pi_3}$. We formulate the one-vehicle CVRP as follows:

$$\min_{\forall x_{i,j}} \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{j=1, j\neq i}^{n+1} x_{ij} = 1, \qquad i = 1, \ldots, n, \tag{2}$$

$$\sum_{i=0, i\neq h}^{n} x_{ih} - \sum_{j=1, j\neq h}^{n+1} x_{hj} = 0, \qquad h = 1, \ldots, n, \tag{3}$$

$$\sum_{j=1}^{n} x_{0j} = 1, \tag{4}$$

$$y_i + q_j x_{ij} - Q^{max}(1 - x_{ij}) \leq y_j, \qquad i,j = 0, \ldots, n+1, \tag{5}$$

$$q_i \leq y_i \leq Q^{max}, \qquad i = 0, \ldots, n+1, \tag{6}$$

$$x_{ij} \in \{0,1\}, \qquad i,j = 0, \ldots, n+1. \tag{7}$$

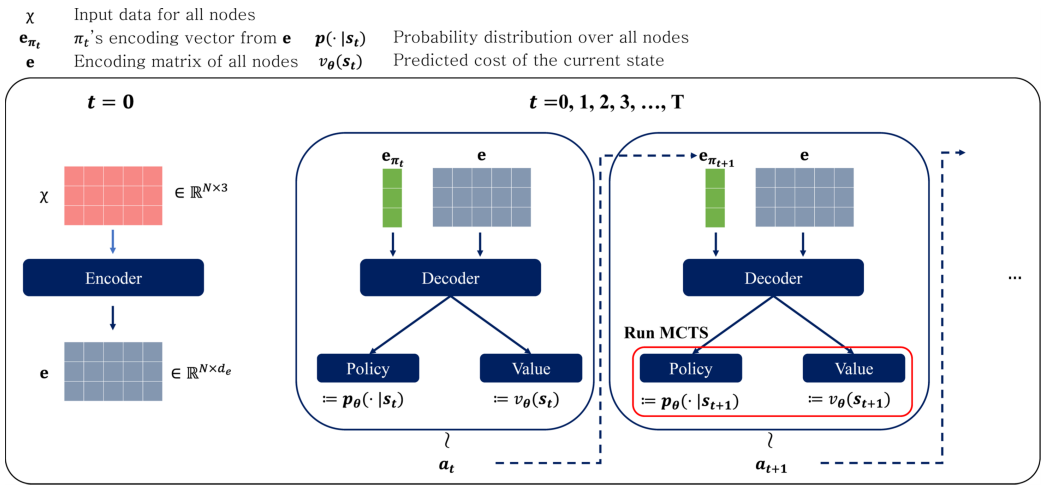


Figure 2: Overall process of routing using our proposed neural networks is shown. It auto-regressively selects the next node. The encoder is executed once per episode, and the decoder is executed at every timestep t.

We briefly explain a list of equations as follows: equation (1) is the objective of the problem, the minimization of the distance traveled by the vehicle; equation (2) is a constraint to regularize all customers being visited only once; equation (3) controls the correct flow of a tour, the visit sequence, by ensuring the number of times a vehicle enters a node is equal to the number of times it leaves the node; equation (4) imposes that only one vehicle leaves the depot; equations (5) and (6) jointly express the condition of vehicle capacity. Note that variants for the constraints are possible, and the main reference to the above formulation is Borcinova [4]. We also notice that the finding of solution $x_{i,j}$ is equivalent to the construction of tour $\boldsymbol{\pi}_{0,T}$: for example, $\pi_1 = 2, \pi_2 = 7$ represent $x_{2,7} = 1$. Noticeably in the formulation, the finding of x_{ij} leads to the construction of y_i. To migrate this formulation into TSP, one only needs to remove constraints regarding the capacity of a vehicle and the demands of customers, so that only decision variable x_{ij} remains.

4 Proposed Network Model, AlphaRouter

In this section, we present our approach, named AlphaRouter, to solving the routing problem using both reinforcement learning and MCTS frameworks. We revise the above routing problem by adding the possibility of refilling the vehicle to reflect realistic situations. We notice that the above routing problem is unable to include the refilling action. We begin by defining the components to bring the environment into our RL problem, followed by neural network models of policy and value. We then outline our idea and implementation to adapt MCTS to the routing problem. Our overall process consists of two stages: training the neural network using reinforcement learning and combining the pre-trained network with the modified MCTS strategy to search for a better solution, meaning tour $\boldsymbol{\pi}_{0,T}$ or $x_{i,j}$ in the CVRP formulation. Due to computational demands associated with the application of MCTS, we adopt a selective application of our MCTS when ambiguity arises in choosing the next customer node that is proposed by the output distribution of the policy network, in which the

output distribution means the distribution of possible next nodes. This selective application enhances computational efficiency while maintaining the effectiveness of the MCTS strategy.

4.1 Reinforcement Learning Formulation

The input is denoted by $\mathbf{x}_i \in \mathbb{R}^2$ which represents a set of coordinates for customer i. The demand of node can be included in the vector if the problem is a type of CVRP: i.e., the input for CVRP is then $[\mathbf{x}_i; q_i] \in \mathbb{R}^3$, where semicolon ; represents a concatenation operation. Also, with n customers, the total number of nodes is $N = n$ for TSP, and $N = n + 1$ for CVRP as one depot node exists. Thus, the input matrix is denoted as $\chi = \mathbf{x} \in \mathbb{R}^{N\times 2}$ for TSP problems, and $\chi = [\mathbf{x}_i; q_i] \in \mathbb{R}^{N\times 3}$ for CVRP.

To bring the problem into a reinforcement learning framework, we define the state, action, and cost (converted to reward). In our work, the observation state at timepoint t, denoted by s_t, is a collection of the node data χ, containing coordinates and demands, the currently positioned node π_t, a set of available nodes to visit, denoted by V_t, and a masking vector for unavailable nodes $m_t \in \mathbf{R}^N$ of which p^{th} element in the vector is filled with 0 if $p \in V_t$ and $-\infty$ if $p \notin V_t$: $s_t = (\chi, \pi_t, V_t, m_t)$. Though masking vector m_t stems from available-node set V_t in our formulation, we intentionally add both m_t and V_t to the state s_t so that the masking vector may be adjusted and redefined reflecting domain requirements just as several masking techniques are possible in Transformer [7, 10, 35] .

We omit t for χ as the node data is invariant over time in this problem: for all time points, χ stays unchanged. However, one could make node data χ varying in time depending on the domain requirement, and the proposed network model is able to handle time-varying χ. For CVRP, the current vehicle's load, denoted by $\text{load}_t = Q^{max} - y_{\pi_t}$, is also added to s_t: $s_t = (\chi, \pi_t, V_t, m_t, \text{load}_t)$. The node set $V_t \subset V$ holds nodes, not visited yet, that are able to fulfill the demands considering load_t.

The action, denoted as a_t, is to choose the next customer node and move to it. The action in an episode, a sequence of possible states, is chosen by our policy neural network, as shown in Figure 2, which outputs a probability distribution over all the nodes given the state at t, s_t. We use $p_\theta(.|s_t)$ to describe the policy network output at time t during the episode rollout. In the training phase, the action is sampled from action distribution $p_\theta(.|s_t)$, $a_t \sim p_\theta(.|s_t)$, as the next node to visit, meaning $\pi_{t+1} = a_t, t \geq 0$, with $\pi_0 = 0$. The sampling operation aims to give the vehicle (or agent) a chance to explore a better solution space in our training phase. In the inference phase, however, we choose the action with the maximum probability, meaning $\hat{a}_t = \text{argmax}_{i\in V_t}\, p_\theta(i|s_t)$ if unvisited nodes exist, and $\hat{a}_t = n + 1$ otherwise.

A value network is designed to predict an overall cost (or distance), in the episode at state s_t. This is later used in updating the MCTS tree's statistics. We will describe in detail how other components work in the next subsection. Specifically, an episode, τ, is a rollout process in which state and action are interleaved over $t = 0, 1, \cdots, T$ until the terminal state s_T is reached: $\tau = (s_0, a_0, C(\boldsymbol{\pi}_{0,0}), \cdots, s_T, a_T, C(\boldsymbol{\pi}_{0,T}))$. In this problem, the terminal state is the state in which all customers are visited and the vehicle has returned to the depot if it is CVRP. Because of possibly

multiple refillings of the vehicle, the last time point T can be different in episodes of CVRP problems. For example, even when the problems have the same size (for example, $n = 50$), the optimal solution path can be different due to different customer locations and demands. Upon reaching the terminal state, no more transitions are made, and the overall distance, $C(\boldsymbol{\pi}_{0,T})$, is calculated.

4.2 Architecture of the Proposed Network Model

The neural network architecture of our policy network for calculating the probability distribution $p_\theta(\cdot|s_t)$ is similar to the one used in previous studies [17, 19]. However, to solve the routing problem, we modified the decoder part, relying on the transformer [35]. We aim to extract meaningful, possibly highly time-dependent and complex, features that are specific to the current state while maintaining the whole node structure. We make the two networks share the same embedding vector, transformed by the current input, s_t at time t. The design of the shared input transformation is a deep-layered network, consisting of an encoder and decoder, to fully take advantage of both the whole node structure and the current node. The structure of the two networks with the shared feature transformation is reminiscent of the architecture from the AlphaGo series [29, 30] and the previously related works [17, 19]. In essence, the input s_t produces the estimated probability of possible next actions via the policy network, $p_\theta(\cdot|s_t)$, and the predicted cost via the value network, $v_\theta(s_t)$. For simplicity, we denote all learnable parameters as θ, actually consisting of parameters from the shared transformation, those from the policy network, and those from the value network.

In detail, we explain the proposed network, dividing it into three parts: encoder in the feature transformation, decoder in the feature transformation, and policy & value. The objective of the encoder is to capture the inter-relationship between nodes. The encoder takes only the node data input χ from the s_t, passing it to a linear layer to align with a new dimensionality, d_e, via the multi-head attention layers, expressed by $MHA(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ with input tensors of query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$. The output of the multi-head attention is an encoding matrix, denoted by $\mathbf{e} \in \mathbb{R}^{N \times d_e}$. Each row vector represents the i^{th} node in the encoding matrix, denoted by $\mathbf{e}_i \in \mathbb{R}^{d_e}$. So, the currently positioned node at time t's encoding is $\mathbf{e}_{\pi_t}$, the embedding vector reflecting the complex and interweaved relationship with the other nodes. In summary, the encoder process is self-attention to the input node data expressed as $\mathbf{e} = MHA(Linear(\chi), Linear(\chi), Linear(\chi))$. This is repeated over several layers in the model. Relying on the idea of hidden states and current inputs in recurrent networks, we execute encoder process once *per episode*, thereby reducing the computational burden, and use the current-node embedding and the current loading load_t as inputs to the decoder in a sequential manner. We provide a detailed explanation later in this section.

The decoder is responsible for revealing the diluted relationships in the encoding matrix $\mathbf{e}$ with additional information if it is given. Specifically, the decoder captures the relationships between the current node χ and the others. For example, let us assume that the vehicle is currently on node i and the current node's embedding is $\mathbf{e}_i$. Notice that we ignore time t in the encoding matrix since it does not change in an episode as the output of the encoder is reused over the episode once it has

been executed. By using this $\mathbf{e}_i$ as the query and the whole encoding matrix $\mathbf{e}$ as key and value, the decoder can reveal the relationships between the current node and the others. When passing the query, key, and value, we apply linear transformations to each of them. One should note that TSP and CVRP have different inputs for the query. In CVRP, the current load, $\text{load}_t \in s_t$ is appended to the query input while TSP is not. While there are several layers for the encoder, we only use one layer of MHA for the decoder. A summarization of the decoder is as follows:

$$\mathbf{d} = MHA(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \in \mathbb{R}^{d_e}, \tag{8}$$

$$\boldsymbol{Q} = \begin{cases} \text{Linear}(\mathbf{e}_{\pi_t}) & \text{for TSP,} \\ \text{Linear}([\mathbf{e}_{\pi_t}; load_t]) & \text{for CVRP,} \end{cases} \tag{9}$$

$$\boldsymbol{K} = \text{Linear}(\mathbf{e}), \quad \boldsymbol{V} = \text{Linear}(\mathbf{e}). \tag{10}$$

The policy layer and value layer are responsible for calculating the final policy $p_\theta(\cdot|s_t)$, a probability distribution on all nodes given s_t, and the predicted distance $v_\theta(s_t)$ output respectively. We compute $p_\theta(\cdot|s_t)$ as follows with a given hyper-parameter C that regulates the clipping:

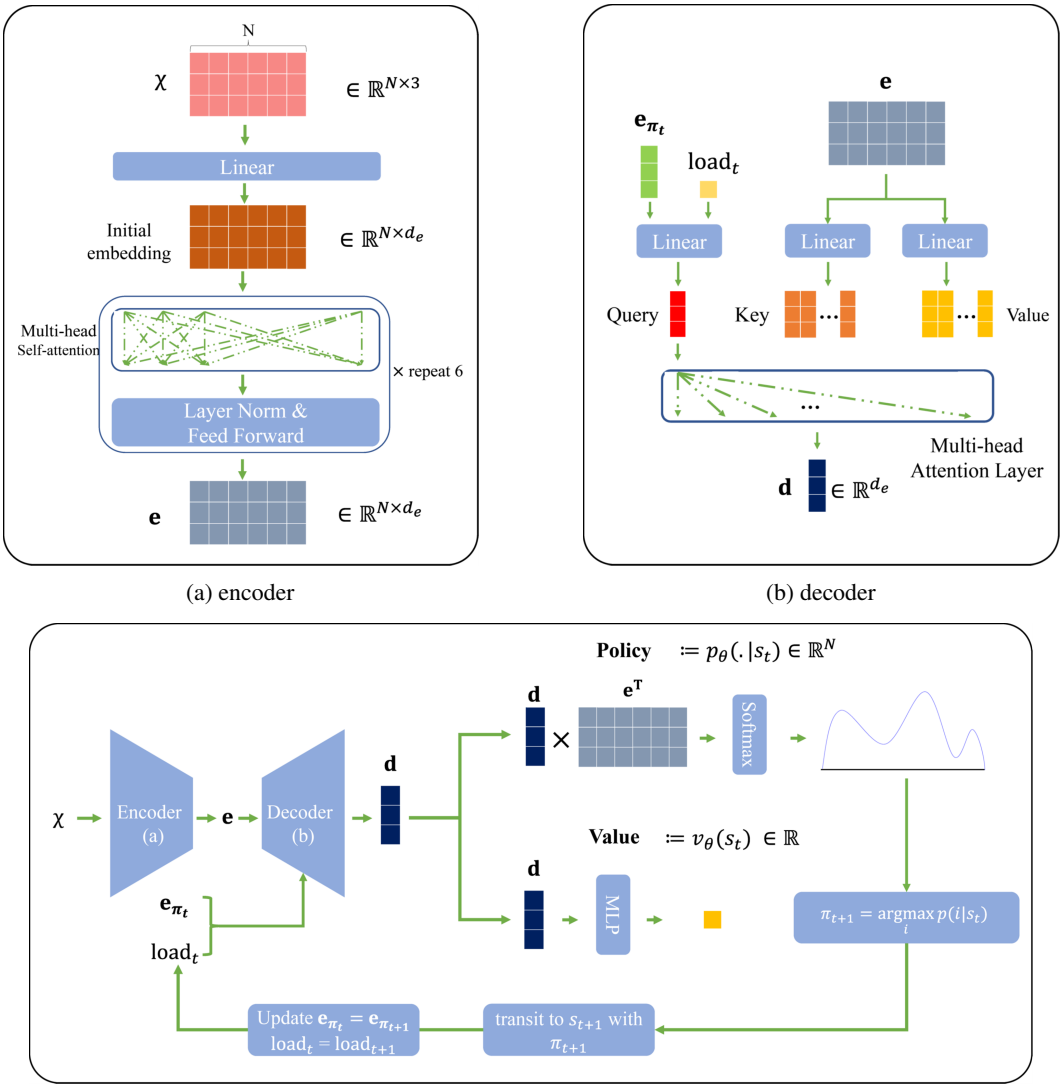
$$p_\theta(\cdot|s_t) = \text{softmax}(\tanh(\mathbf{d}\mathbf{e}^T/\sqrt{d_e})C + m_t). \tag{11}$$

To compute $p_\theta(\cdot|s_t)$, we multiply the decoder output $\mathbf{d}$ by the transposed encoding matrix $\mathbf{e}^T$ and divide it by $\sqrt{d_e}$. The output goes through the tanh function, and we add the mask for the nodes unavailable m_t to it. Finally, we apply a softmax operator to this result.

For $v_\theta(s_t)$, we pass the same decoder output $\mathbf{d}$ to two linear layers of which the shape is similar to the usual feed-forward block in the transformer: $v_\theta(s_t) = \text{Linear}(\sigma(\text{Linear}(\mathbf{d})))$, in which $\sigma(\cdot)$ is an activation function such as ReLU and SwiGLU [1, 28]. A diagram for each neural network design is illustrated in Figure 3.

For training the model for an episode, the encoding process is only required once as the input of the encoder (the coordinates of nodes) is fixed along the rollout steps. The decoder, on the other hand, takes the inputs that change over time, i.e., the current node and current load. Thus, on the first execution of the model, we execute both the encoder and the decoder. After the first execution, we execute only the decoder and policy and value parts, saving considerable computations. Noticeably the encoder and decoder share the same parameters while the policy and value networks do not. Figure 2 explains the overall process.

Additionally, we intentionally exclude residuals in the encoder layers as we have observed that, unlike the original transformer and its variants, residual connections greatly harm the performance of the model. Another variation we have added to the previous is the activation functions. Recent studies on large language models (LLMs) exploited different activation functions for their work. We took this



(a) encoder

(b) decoder

(c) policy & value

Figure 3: Components of the neural network

into account and tested SwiGLU activation, just as Google's PaLM did in [8]. We report the results in later sections.

4.3 Training the Neural Network

To train the policy network θ, we use the well-known policy gradient algorithm, 'reinformce with the baseline'[39]. This algorithm deals with high variance problems prevalent in policy gradient methods by subtracting a specially calculated value, called the baseline. This algorithm collects data during an episode and updates the parameters after each episode ends. For $C(\boldsymbol{\pi}_{0,T})$, the distance traveled by the vehicle following the sequence $\boldsymbol{\pi}_{0,T}$, the policy network aims to learn a stochastic policy that outputs a visit sequence with a small distance over all problem instances. The gradient of the objective function for the policy network is formulated as follows:

$$\nabla J_\theta(\boldsymbol{\pi}) \propto \mathbb{E}_{\boldsymbol{\pi}\sim p_\theta(\cdot|\boldsymbol{s})}[(C(\boldsymbol{\pi}_{0,T}) - b(\boldsymbol{s}))\nabla \log p_\theta(\boldsymbol{\pi}|\boldsymbol{s})], \tag{12}$$

$$\text{where } p_\theta(\boldsymbol{\pi}|\boldsymbol{s}) = p_\theta(\pi_0|s_0) \prod_{k=1}^{T-1} p_\theta(\pi_k|s_k, \pi_{k-1})$$

in which $b(\boldsymbol{s})$ is a deterministic greedy rollout from the best policy trained so far as a baseline to reduce the variance of the original formulation [17]. In detail, after training model parameter θ for an epoch, we evaluated it with a validation problem set, setting $b(\boldsymbol{s})$ as the evaluated cost in the validation. One can think of this procedure as the training-validation mechanism in general machine learning. The mere use of a baseline incurs additional computational costs arising from rollouts of several episodes, being an expensive procedure. To alleviate this burden, we introduced a value network, $b(\boldsymbol{s}) = v_\theta(\boldsymbol{s})$, instead of the greedy rollout baseline.

The value network's objective is to learn the expected cost at the end of the episode from *any state* during the episode rollout. We keep track of the value network's output throughout a rollout and train the network with the loss function

$$L_v \propto \sum_{t=0}^{t=T} (C(\boldsymbol{\pi}_{0,T}) - v_\theta(s_t))^2 \tag{13}$$

As in the POMO approach[19] we tested a baseline using the average cost over a batch of episodes in addition to a baseline using value network $v_\theta(s_t)$. For instance, we calculate the baseline as the mean of all 64 episodes as a batch size, representing the number of concurrent episode runs. This value network is also used in the MCTS process described in the next section. Since our model shares the parameters in the encoder and decoder between the policy network and the value network, an update in the value network affects the parameters in the policy network with the gradient of the final loss as follows:

$$\nabla \mathcal{L} \propto \nabla J_\theta(\boldsymbol{\pi}) + \nabla L_v. \tag{14}$$

4.4 Proposed MCTS for the Routing

The main idea of MCTS is to improve the solutions, good in general, of trained policy and value networks to be problem-specific by investigating possible actions. In essence, without MCTS, we make a transition from s_t to s_{t+1} by taking action a_t, which is the output from the policy network *only*. However, in our proposed MCTS as described in Figure 2, we select the next node by considering costs, which is the output of the value network, in addition to the prior probabilities from the policy network. In addition, we selectively apply the MCTS at time t when the highest probability from the current policy network fails to dominate, meaning actions other than the highest-probability action

need to be considered. In practice, when the difference between the highest probability and the 5^{th} highest probability is less than 0.75, we apply the MCTS, expounded below.

MCTS comprises three distinct phases: selection, expansion, and backpropagation. They iterate with a tree, initialized by the current node π_t and updated as iterations continue, for a given number of simulations, denoted by ns as the total number of the MCTS iterations. At each iteration, the tree keeps expanding, and the statistics of some nodes in the tree are updated. As a result, a different set of tree node paths are explored throughout the MCTS iterations. Figure 4 describes the MCTS procedure in which a few MCTS iterations have been run. Given time t, we use $s_{k|t} = (\chi, \pi_{k|t}, V_{k|t}, m_{k|t})$ to represent a tree node positioned at level k. The definition of $s_{k|t}$ is the same as s_t with only the difference that $s_{k|t}$ represents inner time step k temporarily used in the MCTS selection. Thus, in an MCTS iteration, with fixed t, level k advances as different levels are selected in the selection phase.

In the beginning, we initialize the root tree node $s_{0|t}$ with s_t, meaning the MCTS starts from s_t; thereby the vehicle position in $s_{0|t}$ is the same as the position at t, $\pi_{0|t} = \pi_t$. To describe the MCTS phases, we introduce new notations: for the i^{th} customer (or depot) node, $H^{(i)}_{k|t}$ denotes an accumulated visit count, and $W^{(i)}_{k|t}$ an accumulated total cost, both at the k^{th} level of the tree. Then, we compute the ratio $Q^{(i)}_{k|t} = W^{(i)}_{k|t}/H^{(i)}_{k|t}$, called Q-value. The Q-value, $Q^{(i)}_{k|t}$, for the i node represents an averaged cost at the level k. We normalize all Q-values in the simulation by min-max normalization.

In the selection phase, given the current MCTS tree, we recursively choose child nodes until we reach a leaf node in the tree. For instance, at the k^{th} level of the tree node, among possible nodes, denoted by $V_{k|t}$, we select the next node at $s_{k|t}$ according to equation (15), thereby moving to a tree node at the $k + 1^{th}$ level:

$$\pi_{k+1|t} = \hat{a}_{k|t} = \underset{i \in V_{k|t}}{\operatorname{argmax}} -Q^{(i)}_{k+1|t} + \frac{c_{\text{puct}}\sqrt{H^{(i)}_{k+1|t}}}{1 + \sum_{j \in V_{k|t}} H^{(j)}_{k+1|t}} p_\theta(i|s_{k|t}), \tag{15}$$

in which hyper-parameter c_{puct} adjusts the contribution of the policy-network evaluation $p_\theta(\cdot|s_{k|t})$ in comparison with the negative of averaged cost $Q^{(i)}_{k|t}$ for node i. Let us use ℓ to denote the leaf level in the tree in the selection phase. We obtain an inner *state* path $\boldsymbol{s}_{0,\ell|t} = [s_{0|t}, \cdots, s_{\ell|t}]$ and an inner *node* path $\boldsymbol{\pi}_{0,\ell|t} = [\pi_{0|t}, \pi_{1|t}, \cdots, \pi_{\ell|t}]$. Then the total *node* path from time 0 to the level ℓ becomes a concatenation of $[\boldsymbol{\pi}_{t-1}; \boldsymbol{\pi}_{0:\ell|t}]$. The selection phase continues until no more child node is available to traverse from the currently positioned node, meaning that the node is a leaf node in the tree. In Figure 2, for instance, node 4 was selected, highlighted in the red line, from the root node in the first selection phase, and $\ell = 1$. Note that, in the next MCTS iteration, the selection phase starts again from the root node $s_{0|t}$ again, not from the leaf node selected from the previous iteration.

After the selection phase, the expansion phase starts, updating the MCTS tree by expanding new child nodes in $V_{\ell|t}$ at node $\pi_{\ell|t}$ and moving to the backpropagation phase. Note that in the early stages of the MCTS iterations, the tree may not have expanded enough to select a terminal node, meaning $V_{\ell|t} \neq \emptyset, \ell < T - t$. As the MCTS iteration further advances, the tree expands enough so that the

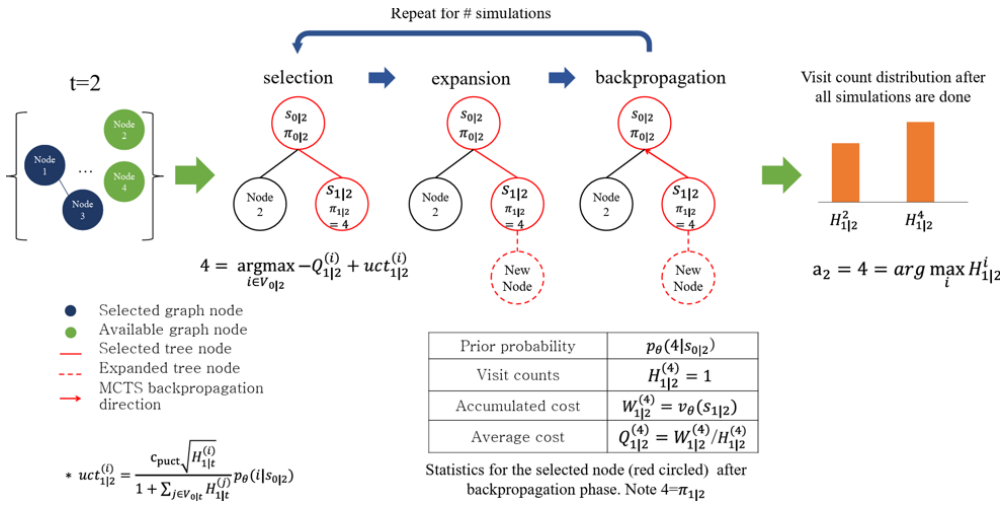


Figure 4: An overall process of transition using MCTS. It depicts the situation when MCTS is run at $t = 2$, and some simulation iterations are done.

final selected node, $\phi_{\ell|t}$, from the selection phase becomes the terminal node, $V_{\ell|t} = \emptyset, \ell = T - t$, meaning that the routing has ended with no available node to move on. In the latter case, the MCTS iteration still continues until it reaches ns in order to explore a variety of possible node paths.

Finally, in the backpropagation phase, tracing back $\boldsymbol{\pi}_{0:\ell|t}$, we update $H^{(i)}_{k|t}$ and $W^{(i)}_{k|t}$ for all selected tree nodes in $k \in [\ell, \ell-1, \cdots, 0]$ and all selected customer nodes $i \in \boldsymbol{\pi}_{0:\ell|t}$. Specifically, the update follows the rule below:

$$H^{(i)}_{k|t} = H^{(i)}_{k|t} + 1, \tag{16}$$

$$W^{(i)}_{k|t} = \begin{cases} W^{(i)}_{k|t} + C([\boldsymbol{\pi}_{0,t-1}; \boldsymbol{\pi}_{0:\ell|t}]), & \ell = T - t, \\ W^{(i)}_{k|t} + v_\theta(\phi_{\ell|t}), & \ell < T - t. \end{cases} \tag{17}$$

As the MCTS iteration continues, the selected leaf node can be either a terminal node ($\ell = T - t$), meaning that the routing has ended, or a non-terminal node ($\ell < T - t$). In the former case, $W^{(i)}_{k|t}$ accumulates the cost by evaluating the selected path of customer nodes, $C([\boldsymbol{\pi}_{0,t-1}; \boldsymbol{\pi}_{0:\ell|t}])$. However, in the latter, we use the predicted distance $v_\theta(\phi_{\ell|t})$. This is possible as we have trained the value network, $v_\theta(\cdot)$, to predict the final distance at any state following (eq. 13). In updating accumulated total cost, $W^{(i)}_{k|t}$, as in equation (17), we obtain the predicted cost by $v_\theta(\phi_{\ell|t})$ at the final selected node $\phi_{\ell|t}$, then greedily selecting the next customer node until the routing finishes.

When finishing all simulations, we collect a visit-count distribution from the $s_{0|t}$'s child nodes and choose the most visited node as a_t for the next node to visit in the rollout:

$$\pi_{t+1} = \hat{a}_t = \underset{i \in V_{0|t}}{\operatorname{argmax}} H^{(i)}_{1|t}. \tag{18}$$

Algorithm 1 Overall simulation flow in MCTS

Require: $s_{0|t}$: root state initialized by s_t, p_θ: trained policy network, v_θ: trained value network, ns: number of simulations to run

Initialize the MCTS tree by $s_{0|t}$
while $i < ns$ **do**
  $\phi_{\ell|t}$, $\boldsymbol{s}_{0:\ell|t}$, $\boldsymbol{\pi}_{0:\ell|t}$ = SELECT($s_{0|t}$) ▷ A leaf node in the MCTS tree is chosen
  EXPAND(p_θ, $\phi_{\ell|t}$) ▷ Expand the MCTS tree from the leaf node using available nodes $V_{\ell|t}$
  if $V_{\ell|t} = \emptyset$ **then** ▷ The selection reached the terminal node
    $c = C([\boldsymbol{\pi}_{0,t-1}; \boldsymbol{\pi}_{0:\ell|t}])$ ▷ (eq. 17)
  else
    $c = v_\theta(\phi_{\ell|t})$ ▷ Use the predicted cost for non-terminal leaf nodes
  end if
  BACKPROPAGATE($\boldsymbol{s}_{0:\ell|t}$, c)
  $i = i + 1$
end while
return $\operatorname{argmax}_{i \in V_{0|t}} H^{(i)}_{1|t}$

Algorithm 1 summarizes the overall process of our MCTS. Additionally, the application of the MCTS is computationally expensive, making it impractical in real-usage. We found that most of $p_\theta(\cdot|s_t)$ outputs have low entropy, meaning the highest probability, $\max_i p_\theta(\cdot|s_t)$, dominates the other values. For example, we show the distribution of entropy values from $p_\theta(\cdot|s_t)$ in Figure 7 of the experiment section. Our idea is that we selectively apply our MCTS to the rollout when $\max_i p_\theta(\cdot|s_t)$ fails to dominate; that is to say when the difference between the highest probability and the fifth highest probability is less than 0.75. We empirically obtained the strategy to improve the solution quality in trading the computation time off.

We present the pseudo-code for each MCTS phase in Algorithm 2. We highlight the modifications made to adapt MCTS to the routing problems. Firstly, we applied min-max normalization to the Q-value calculated during the entire search phase. Since the Q-value ranges in $[0, \infty)$ which is equal to the range of cost (distance), this can cause a computational issue as the term $\frac{c_{puct}\sqrt{H^{(i)}_{k+1|t}}}{1+H_{k+1|t}} p_{\theta(i|s_{k|t})}$ typically falls within range $[0, 1]$. Using a naive Q-value could lead to a heavy reliance on the Q-value when selecting the child node because of the scale difference. To apply min-max normalization to the Q-value in the implementation, we record the maximum and minimum values in the backpropagation phase. Secondly, to minimize the distance, we negate the Q-value so that the search strategy aims to minimize distance. In the pseudo-code, STEP procedure, which we have not included in the paper due to its complexity, accepts the chosen action as input and processes the state to transit to the next state. Internally, we update the current position of the vehicle as the chosen action in addition to the current load of the vehicle if the problem is CVRP. In addition to it, the mask for unavailable nodes, m_t, is updated to prevent the vehicle from moving to visited nodes.

Algorithm 2 List of functions in MCTS

Require: c_{puct}=1.1: hyper-parmeter

function SELECT($s_{0|t}$)
  $node \leftarrow s_{0|t}$
  $\boldsymbol{s}_{0:\ell|t}$ = [$s_{0|t}$]
  $\boldsymbol{\pi}_{0:\ell|t}$ = [$\pi_{0|t}$]
  $k = 0$
  while $node$ has child **do**
    $\pi_{k+1|t} = \mathrm{argmax}_{i \in V_{k|t}} -Q^{(i)}_{k+1|t} + \frac{c\text{puct}\sqrt{H^{(i)}_{k+1|t}}}{1+\sum_{j \in V_{k|t}} H^{(j)}_{k+1|t}} p_\theta(i|s_{k|t})$ ▷ (eq. 15)
    Append $\pi_{k+1|t}$ to $\boldsymbol{\pi}_{0:\ell|t}$
    $node = s_{k+1|t}$ is updated with the child node selected
    Append $node$ to $\boldsymbol{s}_{0:\ell|t}$
    $k = k + 1$
  end while
  $\phi_{\ell|t} = node$ ▷ Also, $\ell = k$
  return $\phi_{\ell|t}$, $\boldsymbol{s}_{0:\ell|t}$, $\boldsymbol{\pi}_{0:\ell|t}$
end function

function EXPAND(p_θ, $\phi_{\ell|t}$)
  for all $i \in V_{\ell|t}$ **do**
    s, $cost$, $done$ = STEP(i, $\phi_{\ell|t}$) ▷ Run STEP with the leaf node's state for the given i
    create a new child node and assign s as the state
    append the child node to $\phi_{\ell|t}$
  end for
end function

function BACKPROPAGATE($\boldsymbol{s}_{0:\ell|t}$, $\boldsymbol{\pi}_{0:\ell|t}$, c)
  get $[\ell, \ell-1, \cdots, 0]$ from $\boldsymbol{s}_{0:\ell|t}$
  for all $k \in [\ell, \ell-1, \cdots, 0]$ and $i \in \boldsymbol{\pi}_{0:\ell|t}$ **do** ▷ k denotes a level from the leaf to the root
    $H^{(i)}_{k|t}$ += 1
    $W^{(i)}_{k|t}$ += c
    $Q^{(i)}_{k|t} = \frac{W^{(i)}_{k|t}}{H^{(i)}_{k|t}}$
    Normalize $Q^{(i)}_{k|t}$
  end for
end function

5 Experiments

At first, we generated problems by constructing N, the number of all nodes, random coordinates of which each coodinate is uniformly distributed in range $[0, 1]$. For CVRP, we assigned the first node as the depot node. In addition, we assigned the demand of each customer, q_i, with an integer between 1 to 10, scaled by 30, 40, 50 for the problem size (n) 20, 50, 100, respectively. We also applied POMO [19] in our training, setting the pomo size as the number of customer nodes. However, in inference, we excluded POMO, as the utilization of MCTS is infeasible. Our implementation, available at the github[2] url below, is built on Pytorch Lightning [34] and Pytorch. [24]. For the setting of MCTS, we set c_{puct} as 1.1, and varied the total number of simulations, ns, by 100, 500, and 1000. We measured the performance on 100 randomly generated problems as described above. For all the tables in this section, 'dist' represents the average distance over all problems, and 'time' represents the average inference time.

For the encoder and decoder settings, the size of each head's dimension is 32 with 4 heads, summed up to the embedding dimension, $d_e = 128$, and 6 encoder layers are used. We trained the model for 300 epochs with batch size 64, and 100000 episodes. Note that this batch size plays as the number of parallel rollouts in training, meaning that 64 episodes were simultaneously executed. We fixed the clipping parameter, C, to 10. We used Adam [16] with learning rate 1e-4, *eps* 1e-7, and *betas* $\in \{0.9, 0.95\}$ without any warm-up or schedulings. For fast training, we used 16-bit mixed precision.

We conducted the experiment on a machine equipped with i5-13600KF CPU, RTX 4080 16GB GPU, and 32GB RAM on Windows 11. For heuristic solvers in the experiment, we used the same machine except with a WSL setting on Ubuntu 22.04.

5.1 Performance Comparison

In this section, we compare the performance of the proposed MCTS-equipped model with that of some heuristics for the two routing problems, TSP and CVRP. The baseline models for TSP are heuristic solvers, LKH3 [13], Concorde [2], Google's OR Tools [12], and Neareset Insertion strategy [25]. These heuristic solvers, developed by optimization experts, serve as benchmarks for assessing optimization capabilities in solving routing problems. For Google's OR Tools, we added a guided-search option, and the result reported here is better than the result reported in previous research [17, 19]. For a fair comparison, the time limit for OR Tools is set to be similar to the longest MCTS runtime, $ns = 1000$, for each n.

For comparison, additionally, we denote the proposed model without the MCTS strategy as an attention model (AM) that leverages solely the proposed neural network without MCTS. When integrating the MCTS strategy with the network model, we varied the number of simulations to investigate its impact on the performance. We evaluated one case using 100 randomly generated by a same generation strategy employed during training. The comparative results of TSP and CVRP

[2]https://github.com/glistering96/AlphaRouter

Table 1: Results of TSP problems

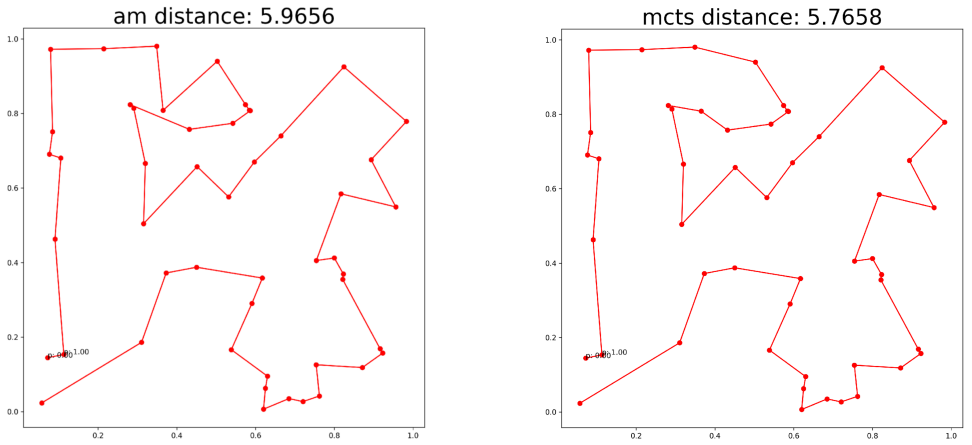
Problem size (n)				20		50		100	
Method	activation	baseline	ns	dist	time	dist	time	dist	time
LKH3	N.A.			3.8402 ± 0.05	0.04± 0.01	5.6705 ± 0.05	0.411 ± 0.07	7.7352 ± 0.05	1.167 ± 0.24
OR Tools				3.8402 ± 0.30	1.001 ± 0.00	5.6807 ± 0.24	1.00 ± 0.00	7.9003 ± 0.28	7.001 ± 0.00
Nearest Insertions				4.3742 ± 0.08	0.001 ± 0.00	6.7550 ± 0.07	0.00 ± 0.00	9.4517 ± 0.08	0.003 ± 0.00
Concorde				3.8402 ± 0.05	0.124 ± 0.03	5.6705 ± 0.05	1.292 ± 0.23	7.7352 ± 0.05	5.143 ± 0.69
DRL	ReLU	mean	0	3.8492 ± 0.09	0.037 ± 0.00	5.7361 ± 0.06	0.079 ± 0.00	7.9852 ± 0.08	0.153 ± 0.00
			100	**3.8486 ± 0.09**	0.043 ± 0.00	5.7375 ± 0.06	0.205 ± 0.01	7.9854 ± 0.08	0.627 ± 0.06
			500	3.8491 ± 0.09	0.059 ± 0.00	5.7345 ± 0.06	0.592 ± 0.23	7.9826 ± 0.08	2.588 ± 1.82
			1000	3.8491 ± 0.09	0.077 ± 0.01	**5.7339 ± 0.06**	1.051 ± 0.85	**7.9807 ± 0.08**	5.509 ± 9.14
		val	0	3.8489 ± 0.08	0.038 ± 0.00	5.7386 ± 0.06	0.078 ± 0.00	8.0481 ± 0.08	0.154 ± 0.00
			100	**3.8483 ± 0.08**	0.038 ± 0.00	5.7310 ± 0.07	0.194 ± 0.01	**8.0465 ± 0.08**	0.648 ± 0.08
			500	**3.8483 ± 0.08**	0.044 ± 0.00	**5.7291 ± 0.07**	0.533 ± 0.21	8.0687 ± 0.09	2.897 ± 2.03
			1000	**3.8483 ± 0.08**	0.052 ± 0.01	5.7301 ± 0.06	0.959 ± 0.88	8.0564 ± 0.08	5.915 ± 9.45
	SwiGLU	mean	0	3.8464 ± 0.08	0.037 ± 0.00	5.7267 ± 0.07	0.080 ± 0.00	7.9562 ± 0.07	0.155 ± 0.00
			100	3.8462 ± 0.08	0.045 ± 0.00	**5.7248 ± 0.07**	0.173 ± 0.01	**7.9523 ± 0.07**	0.594 ± 0.09
			500	**3.8461 ± 0.08**	0.065 ± 0.01	5.7260 ± 0.07	0.458 ± 0.17	7.9530 ± 0.07	2.362 ± 2.72
			1000	**3.8461 ± 0.08**	0.089 ± 0.02	5.7257 ± 0.07	0.799 ± 0.62	7.9539 ± 0.07	4.941 ± 13.23
		val	0	3.8482 ± 0.09	0.038 ± 0.00	5.7405 ± 0.07	0.081 ± 0.00	7.9551 ± 0.07	0.156 ± 0.00
			100	**3.8474 ± 0.09**	0.047 ± 0.00	5.7377 ± 0.07	0.192 ± 0.01	7.9536 ± 0.07	0.646 ± 0.07
			500	3.8478 ± 0.09	0.069 ± 0.01	5.7377 ± 0.07	0.550 ± 0.19	7.9630 ± 0.07	2.781 ± 1.93
			1000	3.8478 ± 0.09	0.093 ± 0.02	**5.7373 ± 0.07**	0.961 ± 0.70	**7.9522 ± 0.07**	5.822 ± 9.99

problems are summarized in Tables 1 and 2, respectively. The column named ns represents the number of simulations in the MCTS strategy, and $ns = 0$ denotes the AM's result without the MCTS strategy. Thus, the 'DRL' method includes the proposed method and the AM method. The column named 'baseline' represents the different baseline $b(s)$ used in (12), in which "mean" represents the mean-over-batches baseline, and 'value' represents the baseline with the value network, v_θ. The baseline term here differs from the heuristic baselines reported in the tables. The best results in the experiment cases of the DRL methods are bold-faced.

Table 2: CVRP problem result

Problem size (n)				20		50		100	
Method	activation	baseline	ns	dist	time	dist	time	dist	time
LKH3	N.A.			6.1528 ± 0.16	3.948 ± 0.46	10.2951 ± 0.24	14.617 ± 1.15	15.4804 ± 0.33	26.03 ± 1.87
OR Tools				6.2049 ± 0.85	1.002 ± 0.00	10.5973 ± 1.25	5.000 ± 0.00	16.273 ± 1.76	18.000 ± 0
DRL	ReLU	mean	0	6.4097 ± 0.75	0.045 ± 0.00	**10.8050 ± 1.65**	0.095 ± 0.00	16.4418 ± 2.99	0.178 ± 0.00
			100	**6.4010 ± 0.76**	0.151 ± 0.01	10.8225 ± 1.59	0.473 ± 0.03	16.4627 ± 3.00	1.326 ± 0.12
			500	6.4065 ± 0.75	0.398 ± 0.13	10.8192 ± 1.60	1.737 ± 0.81	16.4596 ± 2.99	6.652 ± 3.91
			1000	6.4073 ± 0.76	0.685 ± 0.45	10.8087 ± 1.63	3.258 ± 2.95	**16.4387 ± 2.96**	13.633 ± 17.61
		value	0	6.4553 ± 0.80	0.046 ± 0.00	**10.8634 ± 1.65**	0.095 ± 0.00	**16.4463 ± 3.00**	0.177 ± 0.00
			100	6.4452 ± 0.83	0.176 ± 0.01	10.8718 ± 1.61	0.466 ± 0.03	16.4750 ± 3.08	1.409 ± 0.13
			500	**6.4449 ± 0.84**	0.489 ± 0.15	10.8651 ± 1.65	1.773 ± 0.78	16.4494 ± 3.02	7.079 ± 3.34
			1000	6.4479 ± 0.80	0.831 ± 0.47	10.8774 ± 1.67	3.475 ± 3.65	16.4468 ± 2.98	14.504 ± 14.40
	SwiGLU	mean	0	6.4231 ± 0.73	0.046 ± 0.00	10.7940 ± 1.70	0.097 ± 0.00	**16.4043 ± 3.09**	0.180 ± 0.00
			100	**6.4228 ± 0.75**	0.141 ± 0.01	**10.7747 ± 1.71**	0.464 ± 0.05	16.4190 ± 3.10	1.243 ± 0.14
			500	6.4230 ± 0.73	0.346 ± 0.08	10.7878 ± 1.72	1.742 ± 1.02	16.4169 ± 3.10	6.018 ± 4.62
			1000	6.4292 ± 0.74	0.562 ± 0.23	10.7847 ± 1.67	3.260 ± 3.71	16.4117 ± 3.11	12.221 ± 20.71
		val	0	6.4201 ± 0.72	0.046 ± 0.00	10.8402 ± 1.48	0.096 ± 0.00	16.3575 ± 3.04	0.178 ± 0.00
			100	**6.4020 ± 0.74**	0.154 ± 0.01	10.8420 ± 1.53	0.517 ± 0.04	16.3950 ± 2.97	1.188 ± 0.11
			500	6.4055 ± 0.73	0.422 ± 0.10	**10.8336 ± 1.49**	1.959 ± 1.02	**16.3475 ± 3.05**	5.671 ± 3.51
			1000	6.4101 ± 0.72	0.686 ± 0.28	10.8456 ± 1.53	3.683 ± 3.64	16.3492 ± 3.02	11.610 ± 14.31

For TSP, none of the records from AM, denoted by $ns = 0$, is bold-faced, meaning that the MCTS application improved the solution quality. For CVRP, some records using AM are bold but the best records are all from the cases with MCTS. We provide the visualization of the two different methods



(a) Attention only model's result

(b) MCTS applied model's result

Figure 5: Visualization of two methods' resutls on the same test data

results in Figure 5 for better understanding of the effectiveness of MCTS. Visually and quantitatively, the solution in Figure 5(b) by the proposed model is better than that in Figure 5(a) by the AM.

The results reveal that while the application of the MCTS contributes to performance enhancement compared to the ones without the MCTS, it still falls short of the performance achieved by the heuristic models as other research shows [17, 19]. Contrary to our expectation, an increase in the number of simulations does not consistently lead to solution improvement, a decrease in distance. The analysis indicates a lack of a discernible relationship between the number of simulations and the resulting distance. Specifically, for problems with a size of 50, Pearson's correlation coefficient between the two is -0.72, and for the case of CVRP with a size of 100, it is -0.47. In other cases, correlation scores are generally low, below 0.2.

In addition, the MCTS strategy induces a small runtime overhead compared to the method without MCTS, AM. For CVRP problems, the average runtimes of the proposed method are considerably shorter than those of the LKH3 method. However, the runtime increases according to the problem size n. Our explanation is that, as the problem gets bigger, some large problems are hard to solve only by the probability outputs from p_θ, therefore utilizing MCTS more. We argue that to improve the solution quality of AM, numerous samples of solutions, which take a huge amount of time to generate, are required. This result is also shown in the experiment results from [19]. Also, we point out that training the network with more learning epochs to lower a small amount of distance takes quite a long time. For example, to lower about 0.015 in distance by training the network after 300 epochs, we need approximately 24 hours as described in Figure 6 in which the orange line is the regressed line over the observations. However, with MCTS, we can consistently get better solution results within a few seconds and the method is deterministic, unlike sampling methods. Nonetheless, we believe there still is room for improving the runtime of MCTS. The heuristic solvers were written

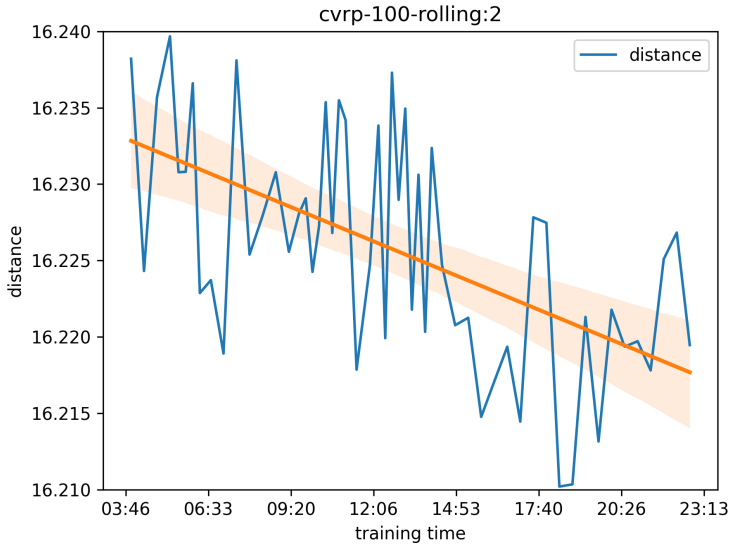


Figure 6: Training score after 300 epochs on CVRP-100 problem smoothed on the window size 2.

in C while our MCTS is written in Python, which is very slow. Implementing MCTS in C++ with parallelism should decrease the runtime much more than now.

Table 3: t-test report within the same conditions

problem type	problem size	AM	MCTS	p-value	<0.05
tsp	20	SwiGLU-val	SwiGLU-val-100	0.1	FALSE
	50	ReLU-val	ReLU-val-500	0.0144	TRUE
	100	ReLU-mean	ReLU-mean-1000	0.0262	TRUE
cvrp	20	SwiGLU-val	SwiGLU-val-1000	0.0066	TRUE
	50	SwiGLU-mean	SwiGLUe-mean-100	0.0994	FALSE
	100	SwiGLU-val	SwiGLU-val-100	0.0339	TRUE

Table 4: t-test report regardless of the conditions

problem type	problem size	AM	MCTS	p-value	<0.05
tsp	20	SwiGLU-val	SwiGLU-val-100	0.1	FALSE
	50	ReLU-val	ReLU-val-500	0.0144	TRUE
	100	ReLU-val	SwiGLU-mean-100	0.0001	TRUE
cvrp	20	SwiGLU-val	SwiGLU-val-1000	0.0066	TRUE
	50	ReLU-val	SwiGLUe-mean-100	0.0064	TRUE
	100	SwiGLU-val	ReLU-mean-100	0.0056	TRUE

To statistically confirm the effectiveness of the proposed MCTS, we include paired t-test results for two different cases. Table 3 reports the test results within the same conditions such as activation function and baselines. The records in the AM column follow the format "activation-baseline" and the records in the MCTS column follow "activation-baseline-MCTS simulation number". In this setting, the test shows that applying the MCTS improves the solution than without the MCTS except for

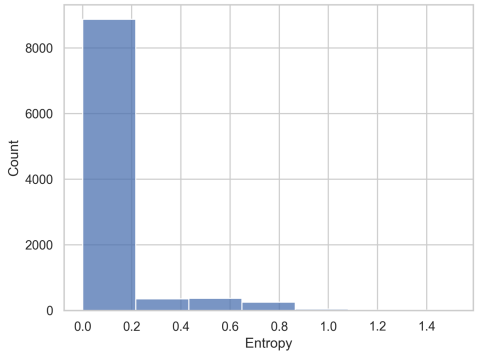


Figure 7: Histogram of entropies of policy network $p_\theta(\cdot)$ for TSP with size 100

TSP-20 and CVRP-50. We suspect that for relatively small size problems, relying on policy network only (AM) can be good enough but if the problem gets bigger, introducing the MCTS can result in better solutions. On the other hand, Table 4 shows the test results regardless of the conditions, so that we report the lowest p-value for each problem type and size. It implies that the application of the MCTS is worth for a given problem type even if the change of activation and baseline is allowed.

We also show the entropy of $p_\theta(\cdot)$ for all test cases in TSP with size 100 in Figure 7. We found that 86% of entropies are below 0.1, meaning that the outcomes of $p_\theta(\cdot)$ are dominated by a few suggestions and a mere application of $p_\theta(\cdot)$ might lead to local optimal. Therefore, while controling time overhead, applying MCTS selectively is a good strategy. We show the evaluation result for not applying MCTS selectively in the Ablation analysis in the next subsection.

5.2 Ablation analysis

In this section, we present some ablation results for the activation function and the baseline. We aggregated the results based on the activation function used and then calculated the mean and standard deviation of the score over all settings, i.e., AM, MCTS-100, MCTS-500, MCTS-1000, and the result is described in Table 5.

Table 5: Performance results according to activation functions

problem	**size**	**20**		**50**		**100**	
type	**activation**	**score**	**runtime**	**score**	**runtime**	**score**	**runtime**
TSP	**ReLU**	3.8487 ± 0.09	0.049 ± 0.00	5.7338 ± 0.06	0.462 ± 0.27	8.0192 ± 0.08	2.312 ± 2.82
	SwiGLU	**3.8470 ± 0.08**	0.060 ± 0.01	**5.7321 ± 0.07**	0.412 ± 0.21	**7.9549 ± 0.07**	2.182 ± 3.50
CVRP	**ReLU**	6.4272 ± 0.79	0.353 ± 0.15	10.8416 ± 1.63	1.421 ± 1.03	16.4526 ± 3.00	5.620 ± 4.94
	SwiGLU	**6.4170 ± 0.73**	0.300 ± 0.09	**10.8128 ± 1.60**	1.477 ± 1.18	**16.3876 ± 3.06**	4.789 ± 5.43

We can easily see that as the problem size gets bigger, SwiGLU scores shorter distances overall than ReLU. Also, we noticed that as the problem size gets bigger, the difference between ReLU and SwiGLU is more apparent. For example, for CVRP problem types, when the problem size is n=20, the difference of the distances is about 0.01 while the number gets bigger n=100, by reaching up to around 0.07. The scalability of SwiGLU is much better than ReLU.

For the baseline, we have suggested two different approaches: mean over batches and value-net-based approach. This baseline is used in (12) as $b(s)$. We calculated the mean and standard deviation of the distances over all settings based on the type of baseline used, like we did in the activation function analysis and the result is described in Table 6. Surprisingly, the mean baseline approach dominates over the value net baseline except for CVRP with problem size 100, which is the hardest problem in the settings. We presume that if the problem becomes complex, the value net baseline may perform better than the mean baseline. For instance, CVRP with a time window and pick-up may be solved better with the value net baseline.

Table 6: Aggregated results of baselines

problem	**size**	**20**		**50**		**100**	
type	**baseline**	**score**	**runtime**	**score**	**runtime**	**score**	**runtime**
TSP	**mean**	**3.8476 ± 0.09**	0.057 ± 0.01	**5.7306 ± 0.06**	0.429 ± 0.24	**7.9687 ± 0.08**	2.116 ± 3.38
	value	3.8481 ± 0.08	0.052 ± 0.00	5.7352 ± 0.07	0.444 ± 0.25	8.0054 ± 0.08	2.378 ± 2.94
CVRP	**mean**	**6.4153 ± 0.75**	0.297 ± 0.11	**10.7996 ± 1.66**	1.391 ± 1.07	16.4319 ± 3.04	5.181 ± 5.89
	value	6.4289 ± 0.77	0.356 ± 0.13	10.8549 ± 1.57	1.508 ± 1.15	**16.4083 ± 3.02**	5.227 ± 4.48

We also report the non-selective MCTS experiment results here. The authors should be aware that the hardware environment here is a little different from the result in the experiment section. For non-selective MCTS, we have run the experiment on a workstation shared among others that runs on Linux with RTX 4090 24GB and Intel Xeon w7-2475X. Therefore, the runtime recorded in the tables below cannot be directly compared to the result from the table from the experiment, but we still include them to observe the trend in the runtime. Despite different hardware settings and computing settings, we still believe that the difference in runtime between selective MCTS and non-selective MCTS is huge enough to support that selective MCTS is meaningful.

Table 7: Non-selective MCTS applied on TSP

Problem size				20		50		100	
Method	activation	baseline	ns	dist	time	dist	time	dist	time
LKH3	N.A.			3.8402 ± 0.05	0.04± 0.01	5.6705 ± 0.05	0.411 ± 0.07	7.7352 ± 0.05	1.167 ± 0.24
OR Tools				3.8402 ± 0.30	5.001 ± 0.00	5.6807 ± 0.24	5.00 ± 0.00	7.9154 ± 0.28	5.001 ± 0.00
Neareset Insertions				4.3742 ± 0.08	0.001 ± 0.00	6.7550 ± 0.07	0.00 ± 0.00	9.4517 ± 0.08	0.003 ± 0.00
Concorde				3.8402 ± 0.05	0.124 ± 0.03	5.6705 ± 0.05	1.292 ± 0.23	7.7352 ± 0.05	5.143 ± 0.69
DRL	ReLU	mean	0	3.8492 ± 0.09	0.107 ± 0.00	5.7361 ± 0.06	0.231 ± 0.00	7.9852 ± 0.08	0.450 ± 0.00
			100	**3.8491 ± 0.09**	0.864 ± 0.03	**5.7356 ± 0.06**	4.614 ± 0.55	**7.9817 ± 0.08**	10.342 ± 2.31
			500	**3.8491 ± 0.09**	3.079 ± 0.25	5.7365 ± 0.06	17.099 ± 2.76	7.9855 ± 0.08	55.007 ± 24.24
			1000	**3.8491 ± 0.09**	5.727 ± 0.34	5.7360 ± 0.06	31.821 ± 5.26	7.9848 ± 0.08	110.943 ± 76.95
		value	0	3.8489 ± 0.08	0.111 ± 0.00	5.7386 ± 0.06	0.230 ± 0.00	8.0481 ± 0.08	0.451 ± 0.00
			100	3.8489 ± 0.08	0.879 ± 0.07	**5.7294 ± 0.07**	4.450 ± 0.60	8.0459 ± 0.08	9.434 ± 3.04
			500	**3.8483 ± 0.08**	2.962 ± 0.08	5.7362 ± 0.06	16.844 ± 3.02	**8.0453 ± 0.08**	56.209 ± 35.65
			1000	**3.8483 ± 0.08**	5.715 ± 0.28	5.7314 ± 0.07	31.341 ± 4.81	8.0456 ± 0.08	112.687 ± 105.85
	SwiGLU	mean	0	3.8464 ± 0.08	0.110 ± 0.00	5.7267 ± 0.07	0.237 ± 0.00	7.9562 ± 0.07	0.454 ± 0.00
			100	3.8463 ± 0.08	0.940 ± 0.13	**5.7263 ± 0.07**	4.292 ± 0.38	**7.9528 ± 0.07**	10.239 ± 3.44
			500	**3.8459 ± 0.08**	2.979 ± 0.12	5.7266 ± 0.07	16.516 ± 1.83	7.9542 ± 0.07	56.392 ± 29.80
			1000	3.8461 ± 0.08	5.716 ± 0.32	5.7265 ± 0.07	30.746 ± 2.69	7.9542 ± 0.07	112.849 ± 84.50
		value	0	3.8482 ± 0.09	0.113 ± 0.00	5.7405 ± 0.07	0.234 ± 0.00	7.9551 ± 0.07	0.456 ± 0.00
			100	**3.8481 ± 0.09**	0.860 ± 0.04	5.7403 ± 0.07	4.429 ± 0.45	**7.9509 ± 0.07**	10.046 ± 3.79
			500	3.8481 ± 0.09	3.040 ± 0.16	**5.7391 ± 0.07**	16.917 ± 2.96	7.9516 ± 0.07	55.425 ± 39.04
			1000	3.8481 ± 0.09	5.705 ± 0.27	**5.7391 ± 0.07**	31.496 ± 4.65	7.9516 ± 0.07	111.637 ± 113.14

We can see that compared to the selective MCTS results from Table 1, a huge amount of runtime is required for MCTS. Also, the bolded record for each group is almost the same or a little bit better with the selective MCTS strategy.

Table 8: Non-selective MCTS applied on CVRP

Problem size				20		50		100	
Method	activation	baseline	ns	dist	time	dist	time	dist	time
LKH3	N.A.			6.1528 ± 0.16	3.948 ± 0.46	10.2951 ± 0.24	14.617 ± 1.15	15.4804 ± 0.33	26.03 ± 1.87
OR Tools				6.2049 ± 0.85	1.002 ± 0.00	10.5973 ± 1.25	5.000 ± 0.00	16.273 ± 1.76	18.000 ± 0
DRL	ReLU	mean	0	6.4097 ± 0.75	0.131 ± 0.00	**10.8050 ± 1.65**	0.279 ± 0.00	16.4418 ± 2.99	0.523 ± 0.00
			100	**6.4048 ± 0.77**	1.067 ± 0.16	10.8133 ± 1.60	3.444 ± 0.57	16.4840 ± 3.06	6.453 ± 2.58
			500	6.4096 ± 0.77	3.258 ± 1.10	10.8100 ± 1.59	14.640 ± 6.72	16.4517 ± 2.99	35.803 ± 47.22
			1000	6.4230 ± 0.79	5.631 ± 2.54	10.8113 ± 1.64	27.046 ± 20.52	**16.4454 ± 3.01**	73.464 ± 156.85
		value	0	6.4553 ± 0.80	0.132 ± 0.00	**10.8634 ± 1.65**	0.276 ± 0.00	**16.4463 ± 3.00**	0.525 ± 0.00
			100	6.4425 ± 0.83	1.185 ± 0.19	10.8717 ± 1.62	3.426 ± 0.50	16.4812 ± 3.08	6.455 ± 2.61
			500	**6.4412 ± 0.82**	3.515 ± 1.18	10.8708 ± 1.66	14.678 ± 6.82	16.4507 ± 3.02	35.825 ± 42.25
			1000	6.4475 ± 0.81	5.890 ± 2.85	10.8715 ± 1.67	27.310 ± 20.23	16.4530 ± 2.98	73.602 ± 166.00
	SwiGLU	mean	0	6.4231 ± 0.73	0.133 ± 0.00	10.7940 ± 1.70	0.278 ± 0.00	**16.4043 ± 3.09**	0.527 ± 0.00
			100	**6.4226 ± 0.75**	1.047 ± 0.15	**10.7908 ± 1.72**	3.371 ± 0.53	16.4193 ± 3.11	6.441 ± 3.31
			500	6.4248 ± 0.73	3.051 ± 0.63	10.7959 ± 1.71	14.491 ± 7.00	16.4164 ± 3.11	35.644 ± 49.24
			1000	6.4333 ± 0.75	5.338 ± 1.48	10.7933 ± 1.69	26.904 ± 22.83	16.4148 ± 3.13	73.034 ± 205.68
		value	0	6.4201 ± 0.72	0.132 ± 0.00	10.8402 ± 1.48	0.277 ± 0.00	16.3575 ± 3.04	0.525 ± 0.00
			100	**6.4047 ± 0.75**	1.129 ± 0.21	10.8389 ± 1.52	3.560 ± 0.66	16.3928 ± 3.01	6.475 ± 2.65
			500	6.4105 ± 0.74	3.314 ± 0.84	**10.8328 ± 1.49**	15.411 ± 7.90	**16.3479 ± 3.04**	35.775 ± 52.90
			1000	6.4138 ± 0.73	5.554 ± 1.49	10.8422 ± 1.51	28.610 ± 27.21	16.3497 ± 3.03	72.945 ± 187.09

We can also observe similar trends on CVRP. Also, one can easily see the huge deviation in runtime for $ns = 1000$ records in both TSP and CVRP. Not only applying MCTS to every transition accounts for this deviation but also the shared resource characteristic of the workstation may have affected it. Therefore, considering the runtime and the performance, selective MCTS is a better approach.

5.3 MCTS application rule

Table 9: TSP 100 difference pivot result

activation	baseline	ns	diff_cut = 0.10		diff_cut = 0.25		diff_cut = 0.50		diff_cut = 0.75	
			score	runtime	score	runtime	score	runtime	score	runtime
SwiGLU	mean	**0**	7.9562 ± 0.07	0.121 ± 0.00	7.9562 ± 0.07	0.121 ± 0.00	7.9562 ± 0.07	0.121 ± 0.00	7.9562 ± 0.07	0.121 ± 0.00
		100	**7.9541 ± 0.07**	0.124 ± 0.00	**7.9541 ± 0.07**	0.124 ± 0.00	7.9541 ± 0.07	0.135 ± 0.00	**7.9523 ± 0.07**	0.621 ± 0.13
		500	**7.9541 ± 0.07**	0.124 ± 0.00	**7.9541 ± 0.07**	0.124 ± 0.00	7.9554 ± 0.07	0.198 ± 0.05	7.9530 ± 0.07	2.586 ± 3.31
		1000	**7.9541 ± 0.07**	0.126 ± 0.00	**7.9541 ± 0.07**	0.126 ± 0.00	**7.9540 ± 0.07**	0.283 ± 0.21	7.9539 ± 0.07	5.553 ± 18.05
	value	**0**	7.9551 ± 0.07	0.156 ± 0.00	7.9551 ± 0.07	0.156 ± 0.00	7.9551 ± 0.07	0.156 ± 0.00	7.9551 ± 0.07	0.156 ± 0.00
		100	**7.9544 ± 0.07**	0.123 ± 0.00	**7.9544 ± 0.07**	0.123 ± 0.00	7.9547 ± 0.07	0.143 ± 0.00	7.9536 ± 0.07	0.679 ± 0.12
		500	**7.9544 ± 0.07**	0.123 ± 0.00	**7.9544 ± 0.07**	0.123 ± 0.00	**7.9527 ± 0.07**	0.244 ± 0.07	7.9630 ± 0.07	3.114 ± 3.52
		1000	**7.9544 ± 0.07**	0.124 ± 0.00	**7.9544 ± 0.07**	0.124 ± 0.00	7.9528 ± 0.07	0.400 ± 0.37	**7.9522 ± 0.07**	6.532 ± 17.42

Table 10: CVRP 100 difference pivot result

activation	baseline	ns	diff_cut = 0.10		diff_cut = 0.25		diff_cut = 0.5		diff_cut = 0.75	
			score	runtime	score	runtime	score	runtime	score	runtime
SwiGLU	mean	**0**	**16.4043 ± 3.09**	0.142 ± 0.00	**16.4043 ± 3.09**	0.142 ± 0.00	**16.4043 ± 3.09**	0.142 ± 0.00	**16.4043 ± 3.09**	0.142 ± 0.00
		100	**16.4043 ± 3.09**	0.351 ± 0.01	**16.4043 ± 3.09**	0.293 ± 0.01	16.4132 ± 3.09	0.213 ± 0.01	16.4190 ± 3.10	1.253 ± 0.15
		500	**16.4043 ± 3.09**	0.329 ± 0.00	**16.4043 ± 3.09**	0.310 ± 0.03	16.4095 ± 3.11	0.528 ± 0.25	16.4169 ± 3.10	6.293 ± 5.36
		1000	**16.4043 ± 3.09**	0.335 ± 0.01	**16.4043 ± 3.09**	0.346 ± 0.16	16.4115 ± 3.12	0.931 ± 1.06	16.4117 ± 3.11	12.700 ± 23.47
	value	**0**	**16.3575 ± 3.04**	0.142 ± 0.00	**16.3575 ± 3.04**	0.142 ± 0.00	16.3575 ± 3.04	0.142 ± 0.00	16.3575 ± 3.04	0.142 ± 0.00
		100	16.3595 ± 3.05	0.341 ± 0.01	16.3597 ± 3.05	0.301 ± 0.01	**16.3536 ± 2.98**	0.217 ± 0.01	16.3950 ± 2.97	1.201 ± 0.12
		500	16.3595 ± 3.05	0.327 ± 0.00	16.3597 ± 3.05	0.318 ± 0.04	16.3549 ± 3.04	0.541 ± 0.27	**16.3475 ± 3.05**	5.947 ± 4.08
		1000	16.3595 ± 3.05	0.343 ± 0.01	16.3597 ± 3.05	0.333 ± 0.10	16.3551 ± 3.04	0.952 ± 1.14	16.3492 ± 3.02	12.072 ± 16.35

In this section, we report two different methods for applying MCTS. The first is the introduced method, which uses the difference between the highest probability and the fifth-highest probability (diff_cut) and the second is to use the entropy of the policy network directly (ent_cut). For each

method, we compare how different pivot numbers affect the performance and runtime. By adopting the result from activation, we only used SwiGLU activation for this test and for better visibility of changes, we focused on the problem with size 100. The difference method's results are reported in 9,10 and the entropy method's results are reported in Tables 11 and 12.

Tables 9 and 10 show the results of different pivot values selected in the MCTS application. For example, diff_cut=0.25 denotes that we apply MCTS when the difference between the highest probability and the fifth-highest probability is less than 0.25, we apply MCTS. If the difference is large, suggestions from p_θ are convincing, and if the difference is small, relying on p_θ only is not enough. Therefore, the higher the diff_cut is, the more MCTS is applied, and vice versa. For both of the problem types, not much difference is observed when the diff_cut value is less than 0.5. However, if the diff_cut is set with values equal to or larget than 0.5, we observed that MCTS does make a better change. We also found that SwiGLU coupled with the value baseline outputs generally producing better solutions than with the mean baseline outputs.

Table 11: Results for TSP 100 with entropy pivots

activation	baseline	ns	ent_cut = 0.25		ent_cut = 0.5		ent_cut = 0.75		ent_cut = 1	
			score	runtime	score	runtime	score	runtime	score	runtime
SwiGLU	mean	**0**	7.9562 ± 0.07	0.123 ± 0.00	7.9562 ± 0.07	0.123 ± 0.00	7.9562 ± 0.07	0.123 ± 0.00	7.9562 ± 0.07	0.123 ± 0.00
		100	**7.9532 ± 0.07**	1.180 ± 0.21	**7.9532 ± 0.07**	0.499 ± 0.04	**7.9541 ± 0.07**	0.358 ± 0.02	**7.9541 ± 0.07**	0.314 ± 0.01
		500	7.9544 ± 0.07	5.739 ± 7.98	7.9542 ± 0.07	1.681 ± 1.16	7.9553 ± 0.07	0.728 ± 0.67	7.9554 ± 0.07	0.496 ± 0.25
		1000	7.9543 ± 0.07	12.532 ± 41.43	7.9543 ± 0.07	3.405 ± 5.95	7.9543 ± 0.07	1.277 ± 3.42	7.9547 ± 0.07	0.729 ± 1.12
	value	**0**	7.9551 ± 0.07	0.124 ± 0.00	7.9551 ± 0.07	0.124 ± 0.00	7.9551 ± 0.07	0.124 ± 0.00	7.9551 ± 0.07	0.124 ± 0.00
		100	7.9537 ± 0.07	1.361 ± 0.25	7.9541 ± 0.07	0.556 ± 0.04	7.9554 ± 0.07	0.394 ± 0.03	7.9547 ± 0.07	0.317 ± 0.01
		500	7.9578 ± 0.07	6.963 ± 8.41	7.9584 ± 0.07	2.081 ± 1.45	**7.9506 ± 0.07**	0.925 ± 0.94	7.9527 ± 0.07	0.576 ± 0.41
		1000	**7.9523 ± 0.07**	15.041 ± 43.92	**7.9529 ± 0.07**	4.243 ± 6.65	7.9514 ± 0.07	1.711 ± 4.56	**7.9525 ± 0.07**	0.867 ± 1.67

Table 12: Results for CVRP 100 with entropy pivots

activation	baseline	ns	ent_cut = 0.25		ent_cut = 0.5		ent_cut = 0.75		ent_cut = 1	
			score	runtime	score	runtime	score	runtime	score	runtime
SwiGLU	mean	**0**	**16.4043 ± 3.09**	0.180 ± 0.00	**16.4043 ± 3.09**	0.180 ± 0.00	**16.4043 ± 3.09**	0.180 ± 0.00	**16.4043 ± 3.09**	0.180 ± 0.00
		100	16.4242 ± 3.09	2.854 ± 0.63	16.4164 ± 3.09	0.910 ± 0.10	16.4145 ± 3.07	0.441 ± 0.08	16.4143 ± 3.08	0.218 ± 0.01
		500	16.4106 ± 3.11	16.915 ± 20.97	16.4118 ± 3.10	4.370 ± 2.99	16.4146 ± 3.11	1.578 ± 2.71	16.4130 ± 3.11	0.547 ± 0.22
		1000	16.4100 ± 3.11	34.895 ± 87.57	16.4100 ± 3.11	8.889 ± 13.80	16.4120 ± 3.12	3.125 ± 11.77	16.4088 ± 3.12	0.974 ± 0.93
	value	**0**	16.3575 ± 3.04	0.178 ± 0.00	16.3575 ± 3.04	0.178 ± 0.00	16.3575 ± 3.04	0.178 ± 0.00	16.3575 ± 3.04	0.178 ± 0.00
		100	16.3839 ± 3.05	2.730 ± 0.61	16.3802 ± 3.02	0.899 ± 0.11	16.3680 ± 3.02	0.459 ± 0.08	16.3666 ± 3.03	0.232 ± 0.01
		500	16.3561 ± 3.06	15.386 ± 15.60	**16.3530 ± 3.05**	4.338 ± 2.80	16.3575 ± 3.06	1.658 ± 1.89	16.3583 ± 3.05	0.647 ± 0.29
		1000	**16.3558 ± 3.04**	31.816 ± 66.79	16.3531 ± 3.03	8.700 ± 11.07	**16.3573 ± 3.04**	3.256 ± 8.14	**16.3572 ± 3.04**	1.168 ± 1.18

Table 11 and 12 show the results of different pivot values selected for the MCTS application using entropy value of p_θ. For example, we only apply MCTS when the entropy of p_θ exceeds the given pivot value. So, the higher the ent_cut is set, the less MCTS is applied. Overall, the performance of ent_cut is not better than the diff_cut method in general. In addition, we can observe that the value baseline generally works better with MCTS than the mean baseline in CVRP. We suspect that the diff_cut method leads to a narrower search boundary than the ent_cut method, which is similar to the trust region in gradient descents.

In summary, heavy applications of MCTS may result in unsatisfactory performance, increaseing runtime quite much. Finding the optimal pivot for the MCTS application may be an important issue. Therefore, we chose the diff_cut method with 0.75 as our final choice in the proposed work.

6 Conclusions and Future Works

We have applied MCTS selectively in routing problems to answer whether it helps generate better solutions. Although the performance was still not enough to reach heuristic solvers, applying MCTS does generate better solutions than the case without MCTS. We also confirmed that using SwiGLU activation rather than typical ReLU can produce better solutions. The result of the baseline experiment is controversial nonetheless using the mean-over-batches baselines helps in generating better solutions generally. We believe that applying MCTS to different VRP problems with more complex situations may reveal the efficacy of a value-network-based baseline.

Some more works can be developed from this paper. Firstly, the runtime of applying MCTS can be reduced if it is implemented in C++, which is much faster than Python. Also if it is implemented in C++, one may try a parallelized version of MCTS [6] to boost the simulation time. Second, MCTS can be extended to other NP-hard problems in the field, i.e., bin-packing, and knap-sack. Third, it is worthwhile to check if MCTS helps generalization across different problem sizes.

References

[1] A. F. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde tsp solver, 03.12.19.

[3] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural Combinatorial Optimization with Reinforcement Learning, Jan. 2017.

[4] Z. Borcinova. Two models of the capacitated vehicle routing problem, 2017.

[5] S. Bubeck and N. Cesa-Bianchi. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems, Nov. 2012.

[6] G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik. Parallel monte-carlo tree search. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 60–71. Springer, 2008.

[7] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1290–1299, 2022.

[8] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[9] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning Combinatorial Optimization Algorithms over Graphs, Feb. 2018.

[10] Y. Du, P. Xie, M. Wang, X. Hu, Z. Zhao, and J. Liu. Full transformer network with masking future for word-level sign language recognition. *Neurocomputing*, 500:115–123, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.05.051. URL `https://www.sciencedirect.com/science/article/pii/S0925231222006178`.

[11] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 10 2022. doi: 10.1038/s41586-022-05172-4.

[12] V. Furnon and L. Perron. Or-tools routing library, 2023. URL `https://developers.google.com/optimization/routing/`.

[13] K. Helsgaun. *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems: Technical report*. Roskilde Universitet, Dec. 2017.

[14] A. Hottung, Y.-D. Kwon, and K. Tierney. Efficient Active Search for Combinatorial Optimization Problems, Mar. 2022.

[15] P. Kilby, P. Prosser, and P. Shaw. Dynamic vrps: A study of scenarios. *University of Strathclyde Technical Report*, 1(11), 1998.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[18] S. N. Kumar and R. Panneerselvam. A Survey on the Vehicle Routing Problem and Its Variants, May 2012.

[19] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min. Pomo: Policy optimization with multiple optima for reinforcement learning, 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/f231f2107df69eab0a3862d50018a9b2-Paper.pdf`.

[20] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34: 5138–5149, 2021.

[21] S. Lin and B. W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2):498–516, 1973. doi: 10.1287/opre.21.2.498.

[22] G. Luo, Y. Wang, H. Zhang, Q. Yuan, and J. Li. Alpharoute: Large-scale coordinated route planning via monte carlo tree search. *Proceedings of the AAAI Conference on Artificial*

Intelligence, 37(10):12058–12067, Jun. 2023. doi: 10.1609/aaai.v37i10.26422. URL https://ojs.aaai.org/index.php/AAAI/article/view/26422.

[23] M.-T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation, Sept. 2015.

[24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[25] D. Rosenkrantz, R. Stearns, and P. II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 09 1977. doi: 10.1137/0206041.

[26] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839):604–609, Dec. 2020. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-020-03051-4.

[27] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[28] N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[29] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, Jan. 2016. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature16961.

[30] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676): 354–359, Oct. 2017. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature24270.

[31] A. Sinha, U. Azad, and H. Singh. Qubit routing using graph neural network aided monte carlo tree search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9935–9943, Jun. 2022. doi: 10.1609/aaai.v36i9.21231. URL https://ojs.aaai.org/index.php/AAAI/article/view/21231.

[32] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks, Dec. 2014.

[33] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[34] T. P. L. team. Pytorch lightning, opensource, 2023. URL https://www.pytorchlightning.ai. https://github.com/Lightning-AI/lightning.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need, Dec. 2017.

[36] T. Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2021.105643. URL https://www.sciencedirect.com/science/article/pii/S030505482100349X.

[37] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60:611–624, 06 2012. doi: 10.1287/opre.1120.1048.

[38] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer Networks, Jan. 2017.

[39] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.