

Pointer Amplifier for Vehicle Routing Problem: Solving Time-Constrained Vehicle Routing Problems with Finite Fleet Using End-to-End Neural Combinatorial Optimization

Elija Deineko¹  and Carina Kehrt¹

Transportation Research Record
1–17

© The Author(s) 2026



Article reuse guidelines:

sagepub.com/journals-permissions

DOI: 10.1177/03611981261485234

journals.sagepub.com/home/trr



Abstract

Finding prompt and optimal solutions to the vehicle routing problem (VRP) is a prerequisite for efficient freight transport, seamless logistics, and sustainable mobility. This requires algorithms that can quickly produce accurate solutions in minimal computational time. Traditional optimization methods struggle with the real-world complexity of VRPs that involve various constraints and different objectives. Recently, advances in neural combinatorial optimization (NCO), which employs generative artificial intelligence to solve combinatorial problems, have demonstrated mature results with promising perspectives. We present an NCO solution method for a time-constrained capacitated VRP with finite vehicle fleet. Our method incorporates customer constraints dynamically, making it flexible and applicable across various transport logistics scenarios. The pointer amplifier, a mechanism introduced here, prioritizes urgent services during tour generation by directly rescaling attention scores in the pointer network. For capacity constraints, we apply reward function penalties with masking to exclude infeasible actions. The architecture uses an encoder-decoder following the Policy Optimization with Multiple Optima protocol and trained via the Proximal Policy Optimization algorithm. The proposed framework was successfully trained and validated for medium and large instances, providing competitive cost-efficient solutions while exhibiting robustness and generalization, compared with state-of-the-art heuristics. Finally, we provide a critical analysis of the solutions generated by NCO and discuss the challenges and opportunities of this new branch of deep learning algorithms.

Keywords

neural combinatorial optimization, vehicle routing problem, reinforcement learning, generative artificial intelligence, deep learning in optimization

Introduction

The vehicle routing problem (VRP) remains one of the most relevant and well-studied combinatorial problems in transportation and optimization science. In practice, however, the formulation of the VRP is subject to many constraints. These include, for example, the need to seek trade-offs between immediate dispatch and later load consolidation, to balance cost minimization with efficient fleet utilization, and to operate under such constraints as limited vehicle availability and delivery time windows (TWs). The development of real-time planning algorithms and decision support systems that can produce prompt and exact solutions incorporating context and handling uncertainties is, therefore, crucial in the field of

transportation (1, 2). Recently, driven by advances in artificial intelligence (AI), a new paradigm in optimization science has emerged: neural combinatorial optimization (NCO). The integration of reinforcement learning (RL) with generative AI models offers a promising perspective to address sequential optimization problems, including the VRP. This study deals with the capacitated VRP (CVRP) with a finite homogeneous vehicle fleet and

¹Institute of Transport Research, German Aerospace Center (DLR), Berlin, Germany

Corresponding Author:

Elija Deineko, elija.deineko@dlr.de

hard time constraints for customer services. In this context, the main contributions of this study are as follows.

- We extend a well-tested NCO framework for combinatorial optimization problems—Reinforcement Learning for Operations Research (RLOR) (3), which adopts the OpenAI Gym environment, Policy Optimization with Multiple Optima (POMO), and Proximal Policy Optimization (PPO) algorithms (4–7). We then apply these frameworks to our problem formulation.
- We design and calibrate the penalty function to address the multiobjective problem, that is, maximizing vehicle fleet utilization while minimizing total fleet mileage.
- A pointer network amplifier (PAMP) method for node selection with time-critical services is introduced, which can integrate hard and soft time constraints on the fly, that is, by rescaling the attention values directly in the tour construction phase.
- We thoroughly examine the solutions and benchmark the proposed method against state-of-the-art heuristics.

State of the Art

Over the past decade or so, researchers in optimization and transportation science have employed various approaches to address combinatorial problems. One line of research is focused on mathematical modeling through Markovian frameworks for sequential decision-making (8–13), while another exploits AI-oriented data-driven methods that leverage deep neural networks and RL to generate feasible solutions in a short computational time (14–16). Through this study, we contribute methodologically to the domain of data-driven end-to-end optimization.

Vinyals et al. (17) introduced a pointer network designed to tackle the traveling salesman problem (TSP) by utilizing an attention mechanism (deep learning architecture) to assign importance to each node. Subsequently, Bello et al. (18) and Nazari et al. (19) advanced this approach by integrating a basic embedding technique and expanding the framework with RL, enabling the solution of more complex combinatorial optimization problems, such as the VRP. The self-attention-based encoder-decoder architecture, commonly referred to as the Transformer and introduced by Vaswani et al. (20), has revolutionized sequential problem-solving, replacing classical recurrent neural networks for sequential token generation. In Deudon et al. (21), the Transformer is adapted specifically for TSPs. Building on this framework, Kool et al. (22) demonstrate

the Transformer's ability of to solve complex combinatorial problems, such as the VRP. In particular, Kool et al. achieve state-of-the-art results by independently encoding depot and customer nodes and incorporating a baseline policy during training. More recent work has increasingly adopted this paradigm, applying it to numerous extensions and variants of routing problem (23, 24).

However, in real-world scenarios, the VRP is subject to many constraints, such as TWs (25–28), and heterogeneous vehicle fleets (29–31). This causes specific challenges for NCO, as there are no standardized methods for incorporating such constraints in the learning process. In practice, constraint handling is often achieved through masking techniques, which prevent the RL agent from executing invalid actions (such as visiting the same node twice in the VRP) (32, 33). Furthermore, the agent's behavior can be guided by means of a carefully designed and tuned reward function (34). In this context, Falkner and Schmidt-Thieme (35) introduce an attention model for VRPs with TWs, in which both nodes and routes are embedded via an encoder-decoder attention mechanism. Li et al. (36) apply a similar architecture to the covering TSP. Li et al. (37) propose an attention-based deep RL framework to address the VRP with a heterogeneous vehicle fleet. Peng et al. (38) pursue a dynamic embedding strategy, wherein subproblem instances are re-encoded after each completed route. In addition, Wang et al. (39) show that modifying the decoder architecture by directly incorporating context information can amplify attention scores in real time, that is, during the training and inference phases. By doing so, the agent is forced to prioritize nearby nodes and reduces the dispersion of attentional scores; this is particularly beneficial for large-scale problems. Chen and Luo (40) utilize a similar concept, enhancing the attention score using the distance matrix. In contrast to Wang et al. (39), they feed the product of queries and keys, combined with the distance matrix, into an additional feedforward network to modify the attention scores. Xin et al. (41) identify two key limitations in current state-of-the-art NCO methods that negatively affect solution quality: (i) insufficient diversity in partial solutions and (ii) the static embedding of the entire graph at the start of each episode. To address the first challenge, the POMO approach introduced by Kwon et al. (7) provides a robust framework for parallel exploration of different node sequences within a single episode, thereby increasing the diversity of partial solutions. The POMO framework relies on trajectories, where each $\text{traj} \in T$ contains the partial dynamic representation of the episode, such as the current vehicle load, dynamic node features, or other episode-specific information. Following this paradigm, Rabecq and Chevrier (42) apply POMO to the pick-up and delivery VRP with

TWs. Fitzpatrick et al. (43) further evaluate the performance and generalization capabilities of POMO for combinatorial problems, demonstrating good solution quality. Wan et al. (3) adapt the POMO approach to RL platforms and introduce a modular framework for solving combinatorial optimization problems—RLOR. Integrated with RL environments such as OpenAI Gym and platforms like CleanRL (44) or RLlib (45), the POMO framework enables concurrent training by learning several distinct solutions simultaneously.

The current state of the art handles VRP constraints through additional embeddings of information or extended decoder architectures. Moreover, to enhance the diversity of partial solutions, more recently, researchers have adopted POMO for combinatorial optimization. In our work, we integrate these two concepts by dynamically rescaling the attention score using environmental information in the decoder and by relying on the RLOR framework with POMO for parallel solution searching.

Methodology

In the following section, the problem formulation will be presented, and the main methodological steps, including the RL environment, the training algorithm, and the policy network architecture will be described. Afterwards, the reward function calibration and the PAMP method will be shown.

Problem Formulation

In this study, we address a CVRP with a homogeneous but finite vehicle fleet and predefined service times for customers. The problem can be formulated as follows. Given a finite fleet of vehicles with identical capacity, the objective is to serve the given demand without violating time and fleet constraints, while minimizing the total mileage performed. The vehicle starts the tour fully loaded at the depot and returns to the depot for reloading if it cannot reach the next customer. The customers are considered as not reachable if their demand exceeds the vehicle's current load or the transport time exceeds the defined service time limit. In fact, we have a multiobjective problem. The objectives are to (i) maximize the vehicle's utilization and (ii) minimize the total mileage performed, subject to TW constraints.

These optimization problems commonly arise in such scenarios as parcel distribution in urban logistics, food distribution, or customer service route planning, where companies with established fleets require efficient routing to provide services within specified business hours. In the following, we will describe the extended problem formulation as well as the design of the RL environment, the

trained AI policy, and the training RL framework design.

Environment Properties

Each of the vehicles $V = \{v_1, v_2, \dots, v_i\}$ must deliver items to N customers (defined as nodes interchangeably). The set of customers $N = \{n_0, n_1, n_2, \dots, n_i\}$ can be divided into visited customers $N^v = \{n_0^v, n_1^v, n_2^v, \dots, n_i^v\}$, visited by vehicle $v_i \in V$, and unvisited customers $N^u = \{n_0^u, n_1^u, n_2^u, \dots, n_i^u\}$, such that $N = N^v \cup N^u$ with n_0 denoting the depot. The objective is to minimize the total route length for all vehicles, maximizing vehicle utilization. Each $n \in \mathbb{R}^4$ is denoted with two-dimensional location coordinates (x, y) , demand d^n , and a time constraint, that is, the end time e^n . At the beginning of each episode (at time step $t = 0$), we assume that the number of vehicles V is given, each with the same capacity C^v . The current load of vehicle v_i at time step t is expressed as l_t^v . When the vehicle's load is zero, it returns to the depot and a new vehicle is deployed, so the number of maximum possible tours is limited by V . It should be mentioned that this problem formulation is dynamic, that is, as time progresses during the tour, the travel time increases. Consequently, this reduces the available TWs of the customers. To address this, the end time e^n (i.e., initial TWs) is reduced at each time step by the travel time covered by the vehicle during the tour.

In the following, we will formalize the problem as a Markov decision process, which is typically defined as a tuple of state s , action a , reward r , and transition Tr: $\text{MDP} = (s, a, r, \text{Tr})$.

State S. The environment state is characterized by the problem state $s_p \in \mathbb{R}^{N \times F}$, a mask $m \in \mathbb{Z}^{\text{traj} \times N}$, and an information state $s_i \in \mathbb{R}^{\text{traj}}$, that is, $S = s_p, m, s_i$. The problem state s_p indicates the initial stationary state randomly generated for each instance. Each node $n \in N$ is described by four features F : two-dimensional coordinates, demand d^n and end time e^n . The mask m consists of binary variables for each node and is updated in each time step t and in each trajectory traj , such that $m^{\text{traj}} \in \{0, 1\}$. The definition of traj refers to the set of parallel state trajectories, as defined for POMO (7). The information state s_i is dynamic and contains information, such as the last node index n_{t-1} , the number of vehicles $V = \{v_1, v_2, \dots, v_i\}$, the current vehicle load l_t^v , and the distance traveled by the deployed vehicle td_t^v .

Action A. The objective in our problem formulation is to construct tours, that is, a sequence of visited nodes for each vehicle v_i , such that $N^{v_i} = \{n_0^{v_i}, n_1^{v_i}, n_2^{v_i}, \dots, n_i^{v_i}\}$. The

Table 1. Sensitivity Analysis of Penalty Factor P for Instances of Three Different Sizes

N	P	Distance (distance units)	Customers (%)	Demand served (%)	Rest load of fleet	Used vehicles
50	1	2.46 ± 0.67	27.6	31.0	2.524	4.46
50	10	8.38 ± 1.26	72.1	80.2	0.012	5.00
50	100	14.25 ± 1.74	79.0	80.3	0.003	5.00
100	1	3.22 ± 0.57	23.1	27.0	1.320	4.64
100	10	6.15 ± 0.87	35.1	40.7	0.003	5.00
100	100	13.02 ± 1.55	47.8	40.8	0.000	5.00
500	1	5.22 ± 1.00	6.5	6.9	0.245	4.56
500	10	4.16 ± 0.71	6.3	8.0	0.001	5.00
500	100	16.79 ± 1.70	14.1	8.0	0.003	5.00

action space is, therefore, the indexes of the visited nodes for each trajectory $a^{\text{traj}} = \{n_t^v\}$.

Transition. The transition function defines the update rules for the current state s_t , after taking the action a_t , that is, when the environment transitions into a new state s_{t+1} . In each state trajectory, the current state components become updated by the following rules.

The equation

$$m_{t+1}^n = \begin{cases} 0, & \forall n^u \text{ if } t = 0 \text{ and } \forall n_0 \text{ if } t > 2 \\ 1, & \forall n^v \text{ if } n_{t+1} = a_t, \text{ or } d_{t+1}^{n^v} > l_t^v, \text{ or } l_t^v = 0, \text{ or } V = 0 \end{cases} \quad (1)$$

expresses that, in initial state $t = 0$ the mask $m_{t=0}^n$ contains zeros for all unvisited nodes n^u , except the depot n_0 , but the depot is unmasked after two steps, that is, $t > 2$. We switch the visited nodes in the mask to 1 if the node was visited, that is, $n^v = a_t$. Additionally, we explicitly mask all nodes where the demand exceeds the current vehicle load: $d_{t+1}^{n^v} > l_t^v$; or, if the remaining load of the vehicle is zero, $l_t^v = 0$; or, if no more vehicles are available, $V = 0$. If the node is not reached and the depot is not masked ($m_{t=0}^{n_0} = 0$), the vehicle returns to the depot and n_0 becomes $m_{t+1}^{n_0} = 1$. Furthermore, in

$$l_{t+1} = \begin{cases} l_t^v - d_{t+1}, & \forall n^v \\ l_t, & \text{otherwise} \end{cases} \quad (2)$$

the load of the current vehicle is reduced by the demand of the visited customer. The equation

$$v_{t+1} = \begin{cases} V - 1, & \text{if } a_t = 0 \\ V, & \text{otherwise} \end{cases} \quad (3)$$

denotes that the number of the vehicles is reduced in each state, if the last action is a depot: $a_t = 0$. The equation

$$td_{t+1}^v = td_t^v + \text{dist}(n_t, n_{t+1}) \quad (4)$$

just gives the cumulative distance traveled by the vehicle fleet V . The equation

$$e_{t+1}^n = e_t^n - td_t^v / vs^v \quad (5)$$

defines the update of the service end time e^n for each customer by subtracting the vehicle travel time, td_t^v / vs^v , where vs^v denotes the constant speed of the vehicle. The problem is terminated when all vehicles are deployed, that is, $V_{t+1} = 0$; therefore, all nodes become masked: $m_{t+1}^n = 1$.

Reward Function. To encourage the agent to visit as many customers as possible (because otherwise the agent will always select the depot as the next node to reduce the total distance), we formulate the reward function as

$$r_t = \min \left(\text{dist}(n_t, n_{t+1}^u) + P \times \sum l_t^v \right) \quad \text{for } t \leq N \quad (6)$$

where dist is the Euclidean distance between the next unvisited node n_{t+1}^u and this node n_t ; l_t^v is the load of vehicle v at time step t . The penalty factor P can be interpreted as an unused vehicle capacity penalty as it penalizes the agent for returning to the depot with the full load $l \rightarrow 1$ and does not penalize the agent when the load of the vehicle $l \rightarrow 0$. To balance the penalty signal and avoid noise in the reward function, the penalty factor P must be carefully calibrated. If the penalty is too low, the agent returns to the depot as soon as possible and will not use the full capacity of the vehicle. If the penalty signal is too high, the reward function will be too noisy and the agent will only seek to exploit the full load at all costs (34, 46). In other words, the penalty factor regulates the relative importance of vehicle utilization compared with distance minimization while satisfying the constraints. In Table 1, we present the sensitivity analysis of the solution quality with respect to the penalty factor P . Here, the results of the sensitivity analysis focusing on the distance, served customers, total served demand, rest load of the

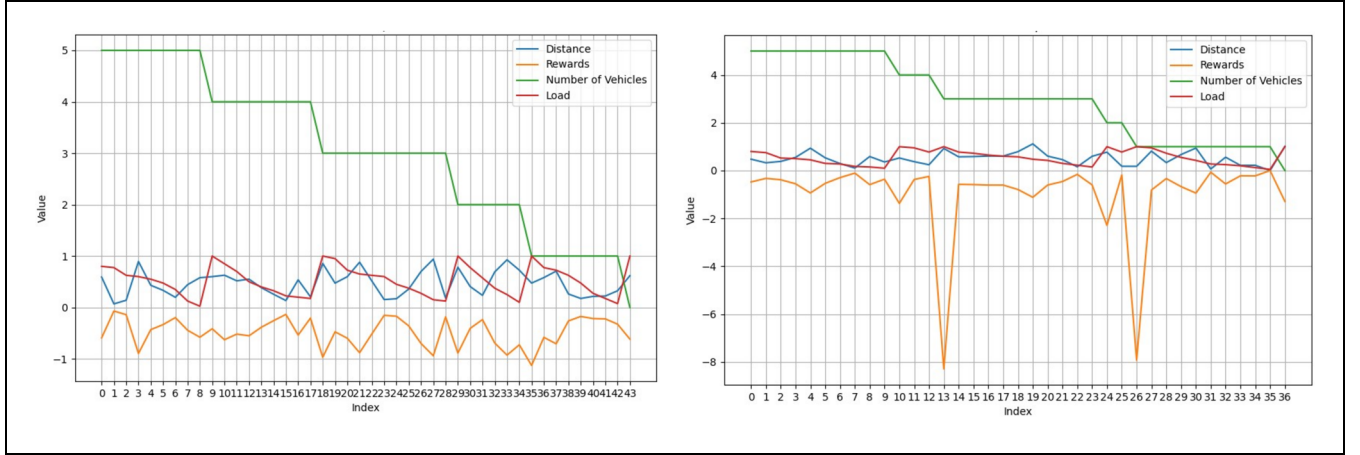


Figure 1. Parameters from the environment, such as distance (blue line), rewards (yellow line), number of vehicles (green line), and current load (red line) during two selected episode rollouts for two episodes with $P = 10$.

vehicle fleet, and the number of utilized vehicles in the obtained solutions for $P = 1$, $P = 10$, and $P = 100$ are shown. The analysis was conducted for three instance sizes ($N = 50, 100, 500$) to characterize the operational implications of the penalty choice across problem scales. For each value of P , 50 instances with different customer distributions were solved for each value of N , and the mean and standard deviation of the resulting solutions are reported.

At $P = 1$, the penalty for unused vehicle load is small relative to the distance term in the reward (Equation 6). The policy is dominated by distance minimization: tours are short, unserved demand is high, fleet utilization is incomplete (4.46–4.64 used vehicles out of 5 on average), and the fraction of customers served is low at every instance size. At $P = 10$, the penalty becomes binding. Fleet utilization reaches the full fleet of five vehicles in all settings and demand served increases at $N = 50$ (31.0% to 80.2%) and $N = 100$ (27.0% to 40.7%). At $N = 500$, the gain in demand served is smaller (6.9% to 8.0%); this indicates that fleet capacity rather than policy behavior is the binding factor at this instance size. At $P = 100$, the penalty dominates the reward. Distance grows by a factor of 1.7 to 4.0 relative to $P = 10$. The marginal gain in demand served over $P = 10$ is small or absent; this is consistent with full fleet utilization.

Figure 1 shows how the environmental components, such as rewards, vehicle loads, number of vehicles and tour length, evolve during the route construction phase for a penalty factor $P = 10$. The decrease in the number of vehicles indicates a return to the depot after t steps. The left panel represents the episode where the vehicle’s load is fully exploited in each tour. The right panel contains tours with unutilized vehicle load. The number of steps t , that is, the number of nodes served by the vehicle, is depicted on the x -axis.

As shown in Figure 1, the reward r varies between 0 and 1 for long tours (Figure 1, left), when the vehicle load is fully exploited. The reward decreases to -8 when the vehicle returns to the depot too early after serving only several customers (Figure 1, right). Thus, the reward decreases proportionally with the distance traveled, ensuring a feasible trade-off. We select $P = 10$ for the main experiments, as it yields the most balanced trade-off between routing distance, served demand, and vehicle fleet utilization.

PAMP Method

Inspired by the distance-aware attention reshaping method introduced by Wang et al. (39), we implement our customer time constraints directly in the decoder; we rescale the attention score in the pointer network by adding an *urgency factor*. Wang et al. (39) adjusted the attention values in the decoder based on the node proximity, resulting in enhanced attention to the nearby nodes within a tour. We use similar logic, but integrate time constraints as the attention rescaling factor, which we call the *urgency factor* for the customers. The urgency factor f_u is calculated as

$$f_u^{n_t} = \frac{\text{dist}(n_t, N^u) / v_s}{e^n - td_t^v / v_s^v}, \quad f_u : \mathbb{R}^{\text{traj} \times N} \quad (7)$$

where f_u expresses the urgency for serving all unvisited nodes N^u for the vehicle being in the current node n_t . In other words, a ratio is computed between expected travel times and end times, which are updated in each time step t . First, we determine the distance between the actual node n_t to all unvisited nodes N^u in this episode and in each trajectory, as $\text{dist}(n_t, N^u)$. We divide these distances by the updated end times for each node (term $e^n - td_t^v$),

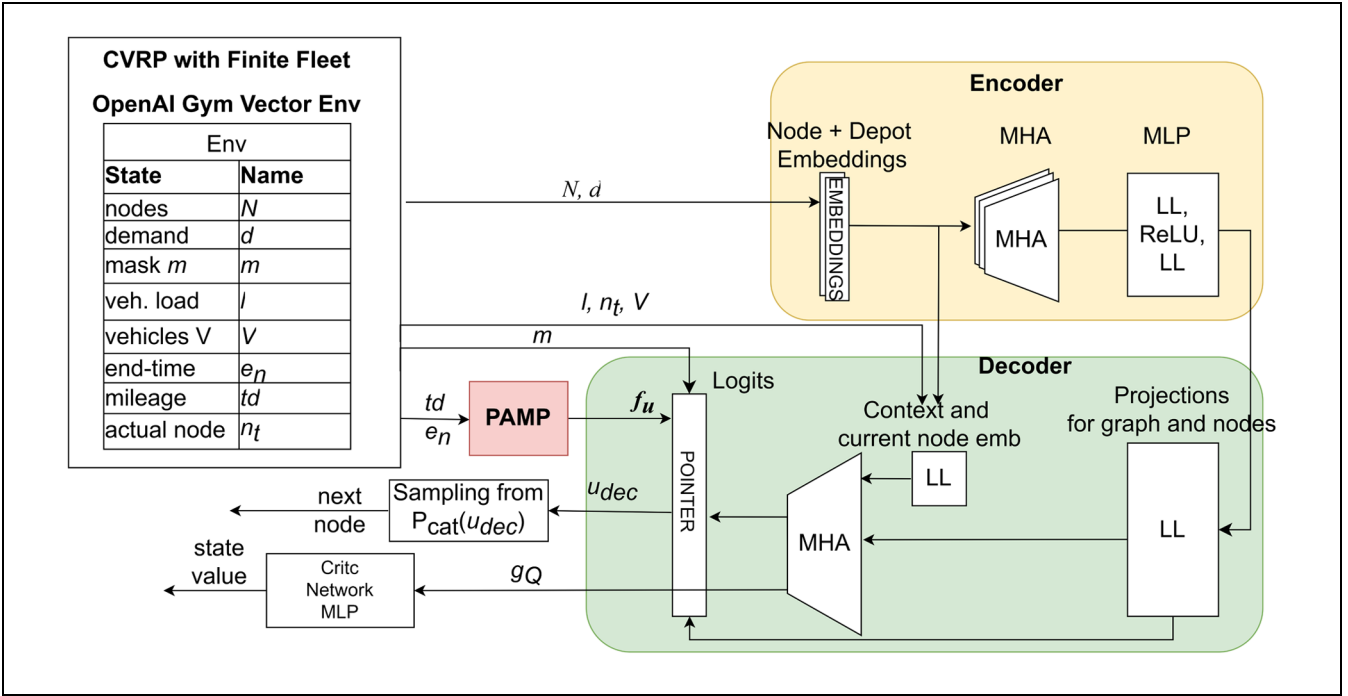


Figure 2. Policy network architecture with its input and output data flows.

Note: AI = artificial intelligence; CVRP = capacitated vehicle routed problem; Env = environment; LL = linear layer; MHA = multihead attention; MLP = multilayer perceptron; PAMP = pointer network amplifier; ReLU = rectified linear unit.

where e^n are given end times for each customer and td_i^v/vs^v is the duration of the vehicle's trip. Again, vs^v represents the vehicle velocity and is needed to convert the distances to travel times. Consequently, we assume that $vs^v = 50\text{km/h} = 0.0014\text{ distance units/s}$ for the normalized instances $x, y \in [0, 1]$. Therefore, if $f_u \rightarrow 0$, the end time requested by the customer lies ahead and there is no need to urge. If $f_u \rightarrow 1$, the service becomes time critical. For $f_u > 1$ or $f_u < 0$, the service is considered as missed and masking is triggered. In other words, we set this customer to $m_t^n = 1 \forall n$ where $0 < f_u < 1$, effectively masking the node to be selected.

Policy Network

The implementation of our policy network is based on the work of Wan et al. (3), who adapt the well-tested model of Kool et al. (22) and extend it to the trainable framework CleanRL with POMO (44).

The policy network design implemented by Wan et al. (3) is an open-source encoder-decoder model with an eight-head self-attention network (multihead attention [MHA]). First, the problem state s_P is embedded through a linear neural network (LNN): $\text{emb}_n = nW_l^{\text{dim}} + b^{\text{dim}}$. Here, the size of the network is determined by the weight matrix W of dimension $\text{dim} = 128$. By concatenating the depot and the node embeddings, we obtain an embedded representation for the problem state across each batch B :

$\text{emb}_{s_P} \in \mathbb{R}^{B \times N \times 128}$. Note that we specifically embed only the static components of the problem state $s_P \in \mathbb{R}^{N \times 3}$; these include the node's coordinates and demand values, and are independent of the trajectories traj . The policy network consists of the *encoder* and the *decoder* (Figure 2).

Encoder. The encoder MHA contains three layers $l^{1,2,3}$, which consume the embedding tensors emb_{s_P} and calculate queries Q , keys K , and values V as linear projections:

$$Q_l = \text{emb}_l W_l^Q, \quad K_l = \text{emb}_l W_l^K, \quad V_l = \text{emb}_l W_l^V \quad (8)$$

with $Q_l, K_l, V_l \in \mathbb{R}^{B \times N \times 128}$ and W_l^Q, W_l^K, W_l^V being trainable parameters for each layer l . We decompose the projected Q_l, K_l , and V_l into $Q_{l,H}^{1,2,3} \in \mathbb{R}^{H \times B \times N \times 16}$, $K_{l,H}^{1,2,3} \in \mathbb{R}^{H \times B \times N \times 16}$, $V_{l,H}^{1,2,3} \in \mathbb{R}^{H \times B \times N \times 16}$, where $H = 8$ is the number of heads. The attention score u_H is calculated as

$$u_H = \text{softmax}\left(\frac{Q_{l,H} K_{l,H}^T}{\sqrt{\text{dim}_K}}\right) V_{l,H}, \quad (9)$$

where u_H becomes $u_H \in \mathbb{R}^{H \times B \times N \times 16}$

$$u_{\text{emb}}^{\text{MHA}} = \text{Concat}(u_H) W_l^h, \text{ resulting in } u_{\text{emb}} \in \mathbb{R}^{B \times N \times 128} \quad (10)$$

The MHA output $u_{\text{emb}}^{\text{MHA}}$ is then sequentially passed through several normalization, linear, and activation layers for each l . The overall calculation procedure is summarized as

$$h_l^1 = \text{BN}(u_{\text{emb}}), \quad (11)$$

where BN is a batch normalization layer

$$h_l^2 = \text{FF}_l^2(\text{ReLU}(h_l^1 * \text{LL}_i^{128 \times 512}), \text{LL}_j^{512 \times 128}), \quad (12)$$

where FF_l^2 is a sequential feed forward network, with LL_i and LL_j being two linear layers. After applying a ReLU activation layer in Equation 12, and another BN in

$$h_l^3 = \text{BN}(h_l^2) \quad (13)$$

the encoder produces a hidden state h_{enc} of size $h_{\text{enc}} \in \mathbb{R}^{B \times N \times 128}$ for each node in N , after processing through three MHA layers $l_{1,2,3}$:

$$h_{\text{enc}} = \text{FF}_{\text{enc}}(h_{l=1}^3, h_{l=2}^3, h_{l=3}^3) \quad (14)$$

Decoder. The decoder consumes the input from the encoder networks h_{enc} , along with additional state information s_i (i.e., the mask and the problem context), and produces a probability distribution over all reachable nodes for each trajectory traj . Typically, the decoder consists of several linear layers for graph and context projections, an MHA network, and an attention score function called a pointer network, which produces the final output (Figure 2).

To obtain a more complete graphical representation of the initial problem instance (a global graph context vector), we first average the hidden representations h_{enc} produced by the encoder (Equation 14) over all nodes N :

$$c_h = \frac{1}{N} \sum_{i=0}^N h_{\text{enc}} \quad (15)$$

resulting in $c_h \in \mathbb{R}^{B \times 128}$. The problem context, that is, the vehicle load and the number of the remained vehicles in each state trajectory, is concatenated in the context state, $c_{\text{state}} = \text{concat}(l^v, V)$, and then concatenated again with the current node embedding:

$$c_o = \text{concat}(\text{emb}_{n_i}, c_{\text{state}}), \quad \text{resulting in } c_o \in \mathbb{R}^{B \times \text{traj} \times 130} \quad (16)$$

This step effectively combines relevant context information, the vehicle load, and the vehicle fleet information, with the hidden representation of the last visited node. The resulting context vector c_o is then passed through a linear layer, again resulting in an output: $c_o \in \mathbb{R}^{B \times \text{traj} \times 128}$. The query q_m for further attention

score calculation is obtained by adding the graph context c_h to the state context c_o , as

$$q_m = c_h + c_o, \quad \text{resulting in } q_m \in \mathbb{R}^{B \times \text{traj} \times 128} \quad (17)$$

This query q_m , together with the keys and values g_k and g_v , and the mask $m \in \mathbb{R}^{B \times \text{traj} \times N}$, are passed to another instance of the MHA. Finally, to compute the probability distribution over all candidate nodes N in each trajectory traj and batch B , attention score calculations are made within the pointer network. The pointer network computes the attention factor u_{dec} as the dot product between the current node and all other embedded nodes:

$$u_{\text{dec}} = \left(\tanh \left[\frac{g_Q * k_L^T}{\sqrt{\text{dim}_{k_L}}} * m \right] + f_u \right) * C, \quad (18)$$

resulting in $u_{\text{dec}} \in \mathbb{R}^{B \times \text{traj} \times N}$

Here, we inject additional time and distance-related information (e^n and td) and calculate the urgency factor f_u as given by Equation 7. In Equation 18, g_Q refers to the state query, and k_L represents the projected keys for each node. The $\tanh(u)$ function rescales the attention scores u in the range of $-1 < u < +1$. Next, f_u is added to augment the attention scores for nodes that need to be served urgently ($f_u \rightarrow 1$). If the service urgency is small ($f_u \rightarrow 0$), we simply rely on the attention values provided by the decoder. Nodes that are either masked by m or unreachable, that is, owing to the expired end time e^n (if $f_u > 1$), are assigned to $-\text{inf}$. We then apply a clipping factor $C = 10$ to rescale the overall attention score in the range $-10 < u_{\text{dec}} < 10$. The next action a is then sampled from the categorical probability distribution P_{cat} , ensuring that the agent learns the stochastic policy:

$$a = P_{\text{cat}}(u_{\text{dec}}), \quad \text{where } a \in \mathbb{R}^{B \times \text{traj}} \quad (19)$$

Lastly, the state value v_{critic} is produced by the critic network:

$$v_{\text{critic}} = \text{MLP}_{\text{critic}}(g_Q) \quad (20)$$

where $\text{MLP}_{\text{critic}}$ consists of two fully connected layers, LL_1^c and LL_2^c , which sequentially process the input tensor g_Q and output a scalar state value v_{critic} : $\text{LL}_1^c = g_Q \mathcal{W}_c^{\text{dim}_{g_Q} \times 128} + b_c^{128}$ and $\text{LL}_2^c = x \mathcal{W}_c^{128 \times 1} + b_c^1$.

The general architecture of the described policy network is illustrated in Figure 2. First, the static problem state is processed through an MHA encoder network, as given by Equations 11 to 14. Next, the hidden problem representation from the encoder c_h , together with the state information from the environment (load l^v , current node, and number of remained vehicles V), is added to the context vector and supplied to the decoder. The dynamic time constraints are converted to the urgency

Algorithm 1. Implementation of Proximal Policy Optimization Training Algorithm 1

1. Collect observations stepping the vectorized environments in parallel.
Initialize policy π_1 .

For U global updates:

For each step $t \in T$, where T is a given number of steps in the env, do:

Sample observations from the env.

Feed into the policy network π_1 (obs).

Return: next action a_{π_1} , action probability $P(a_{\pi_1})$ and the next value v_{π_1} .

Bootstrap next state values by reverse computation from the end of each episode

starting with the last step t_{T-1} for every batch $b \in B$:

Compute advantages using generalized advantage estimation for policy π_1

if episode is not :

 $A_t = \delta_t + \gamma * \lambda * A_{t-1}$, where γ and λ are given coefficients,

and $\delta_t = r_t + \gamma v_{\pi_1(t+1)} - v_{\pi_1(t)}$

else: $A_t = r_t - v_{\pi_1(t)}$

Compute the return from π_1 : $R_{\pi_1} = A_t + v_{\pi_1}$, where

 $R_{\pi_1} \in \mathbb{R}^{T \times B \times \text{traj}}$
2. Update policy network

Flatten collected actions a_{π_1} , action probabilities $P(a_{\pi_1})$, next value v_{π_1} , R_{π_1}

and A_{π_1} .

Form minibatches of the size $b_x = \mathbb{R}^{T \times \frac{B}{n} \times \text{traj}}$, where x indicates shuffled batch

indices; T defines the number of steps in the environment; n is a number of given minibatches, and B is a total number of batches.

For each epoch update policy π_2 :

For each minibatch b_x do:

Sample b_x observations from the policy π_2 :

 $\text{obs}_{b_x, \pi_2} = \{s_{\pi_2}, a_{\pi_2}, P(a_{\pi_2}), v_{\pi_2}, e_{\pi_2}\}$

Calculate a ratio between the old action probabilities under π_1

and the new action probabilities under π_2 as:

 $L_r = \exp(P(a_{\pi_1}) - P(a_{\pi_2}))$

Calculate policy loss: $L_p = \text{mean}(L_r * (-A_{\pi_1}))$, where L_p is a mean policy loss

Calculate value losses: $L_v = \text{mean}(0.5 * (v_{\pi_2} - v_{\pi_1})^2)$, where L_v is a mean value loss

Compute average entropy from π_2 : $e_l = \text{mean}(e_{\pi_2})$

Calculate total loss from policy losses, entropy values and value losses:

 $L_{\text{total}} = L_p - E^{\text{coef}} * e_l + L_v * V^{\text{coef}}$, where E^{coef} and V^{coef} are given coefficients

Perform backpropagation and update policy π_2 using L_{total} .

Assign updated policy $\pi_1 = \pi_2$ and repeat the training for U time steps

factor f_{li} , given by Equation 7, and, together with the mask m , are then passed to the pointer network, prioritizing nodes that require immediate service.

RL Training

In the following section, we describe the training algorithm for the policy network. We collect batches of size B for synchronized parallel vector environments, where each batch contains the state trajectories traj . At each iteration, we use these batches and their trajectories in accordance with the PPO algorithm to update the network weights. Our training procedure follows the PPO implementation described elsewhere (6, 44), which was further adopted and extended to NCO problems by Wan et al. (3). The pseudocode with the training procedure is presented in Algorithm 1.

We start by sampling each episode stepping through n parallel, independent environments and forming batches B . In the first phase of the applied training algorithm (Algorithm 1), we step the environment 100 times (i.e., collecting 100 observations) in $n = 1,024$ parallel environments, resulting in a total of 102,400 batches with 50 trajectories generated for training. In the second phase, we flatten the batch B and split it into eight minibatches $b_x \in B$, each containing 128 episodes. In each training epoch, we shuffle the samples and compute policy losses, entropy values, and value losses. Finally, the total policy loss is used to update the policy π_2 . These and other hyperparameters, such as the learning rate γ and entropy coefficient E^{coef} , are set analogous to the CleanRL protocol (44). The policy is trained for $U = 8,000$ updates in total for 102,400 episodes. For detailed information on implementation and hyperparameter selection, we refer to Wan et al. (3).

Evaluation and Validation

The following describes how the proposed framework is trained; the framework is then validated through an in-depth examination of the inferred solution and benchmarking against state-of-the-art heuristics for VRP.

Evaluation Instances

We generate the training and evaluation problem instances for each episode, following a procedure analogous to comparable state-of-the-art studies, such as Wan et al. (3), Kool et al. (22), and Berto et al. (47). Each parallel environment is initialized with 50 customer locations and a depot, randomly distributed in the normalized Euclidean space (i.e., $0 \leq x, y \leq 1$), resulting in $N = 51$. The node features are then extended by the demand and end time values. Each

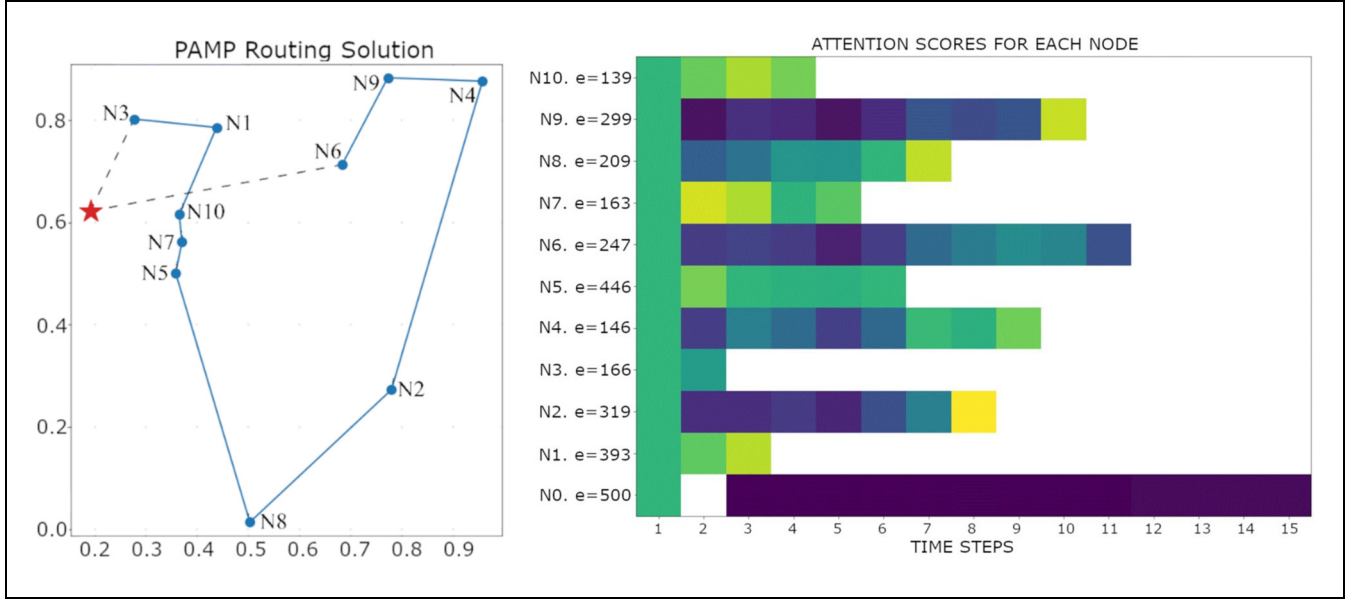


Figure 3. Detailed analysis of solution generated by pointer network amplifier (PAMP) method, for $N = 10$.

problem instance is parametrized by the vehicle capacity $C^v = 40$, which is then normalized to $l_{t=0}^v = 1$. The demand is distributed uniformly: $d_n \sim P_u(1, 10)$ for $n = 1, 2, \dots, N$ and normalized by d^n/C^v . This ensures that the customer demand is always smaller than the vehicle capacity and the minimum tour length for the vehicle v_i becomes $C^v/\max d^n \rightarrow 4$ customers, when operating at full capacity. The total number of vehicles in each instance is set to $V = 5$. Service end time constraints e^n are expressed in time units and are randomly sampled from the range $50 \leq e^{n \neq 0} \leq 10,000$ for customers, and from the range $e^{n=0} = 10,000$ for the depot. This range defines the minimum and maximum travel times for a vehicle traveling at 50 km/h in a normalized instance. This constraint ensures that the most customers are reachable within the reasonable time span.

Evaluation Results

To assess the quality of the decisions produced by the trained model, we initially focus on deploying a greedy policy inference on small instances and showcase the pointer output. The model is trained for 10,000 update steps on the datasets specified in the previous section. The inference is performed on a single instance with $N = 10$ customers and end times $50 \leq e^n \leq 500$. According to Equation 18, the urgency factor f_u , defined as the ratio between the time traveled to the customers and the given end times e^n , is added to the attention score in the pointer network. This effectively highlights urgent services resulting in the higher attention scores for customers with

$f_u \rightarrow 1$. If the customer service end time in time step $t + 1$ lies far ahead (i.e., $e_{t+1}^n \sim 500$) and the distance from this node n_t to the next unvisited node n_{t+1}^u is short (i.e., $\text{dist}(n_t, n_{t+1}^u) \rightarrow 0$), the attention score in the pointer network remain unchanged. Figure 3 (left panel) shows the tour constructed by the trained model for a small test instance with 10 nodes. The solution is generated by the trained PAMP model. For this tour, the total mileage is 3.06 distance units. Figure 3 (right panel) presents the attention score distribution (logit values) for each node (y -axis) and for each step (on the x -axis). In this diagram, the decoder's output, obtained by Equation 18 for each environment step (x -axis) and each node (y -axis), is plotted. The different colors along the y -axis represent the attention score values for each time step and each node.

In the initial time step, the agent starts at the depot, and all attention scores are zero, as shown in Figure 3 (right). The first node, that is, node 3 (Figure 3, left), has the third smallest service end time ($e^{n=10} = 139 < e^{n=7} = 163 < e^{n=3} = 166$). Because it has the shortest distance and the shortest end time, the agent assigns the highest attention score to node N_7 (see Figure 3, right, for $e^{N_7} = 163$). However, the first starting node is N_3 , which is the second-best option when following the greedy policy. Since we are in POMO, the agent builds all 10 trajectories in the initial step and considers all 10 nodes as equally suitable starting points. By doing so, we effectively overcome the issue of starting with the node that leads to the poor solution, even if its attention score was the highest in the initial step. In time step 2, the agent selects node N_1 , probably as it is at the closest

Table 2. Solutions from Benchmark Heuristics and Proposed Pointer Network Amplifier (PAMP) Method

Instances			Solution												
			Jsprit				PyVRP				PAMP (proposed model)				
			<i>d</i>	<i>N</i>	FU (%)	<i>t</i> (s)	<i>d</i>	<i>N</i>	FU (%)	<i>t</i> (s)*	<i>d</i>	<i>N</i>	FU (%)	<i>t</i> (s) CPU	<i>t</i> (s) GPU
N50a	50	1234	5.06	24	69.5	3.7	7.42	44	100	9.5	6.66	33	100	0.3	0.2
N50b	50	4321	6.86	17	49.0	2.3	8.42	39	100	9.7	8.38	32	100	0.3	0.2
N100a	100	1234	6.24	41	91.5	7.4	7.77	63	100	11.0	6.59	41	100	0.5	0.2
N100b	100	4321	6.58	39	81.5	7.2	6.58	57	100	11.3	6.80	40	100	0.5	0.2
N500a	500	1234	6.17	66	98.5	45.7	2.12	43	100	8.8	3.15	27	100	1.9	0.5
N500b	500	4321	6.82	65	99.5	48.0	3.93	67	100	14.5	4.22	33	100	1.9	0.5
N1000a	1,000	1234	5.48	85	100	89.0	2.90	75	100	34.9	3.81	29	100	7.4	1.6
N1000c	1,000	3241	6.58	87	99.5	71.5	3.71	83	100	51.7	4.53	27	100	8.2	1.6

Note: CPU = central processing unit; GPU = graphics processing unit; FU = fleet utilization.

*Only solution time measured; calibration (one-time per instance) excluded.

distance, as shown in Figure 3 (left panel). In step $t = 3$, the node N_{10} exhibits the highest attention value, owing to its narrow end time $e^n = 10 = 139$ and the shortest distance to N_1 . As illustrated in Figure 3, all remaining nodes are selected according to the greedy policy, that is, following their highest attention scores. As a result, all 10 customers are visited without violating the time constraints, forming a reasonable and consistent tour.

Validation

To evaluate the solution quality and generalization capability of our method, we benchmark PAMP against two state-of-the-art heuristics for VRP: PyVRP (48) and Jsprit (49). Both are open-source solvers for multiconstrained VRPs and can be extended to various routing problems, including the CVRP and VRPs with TWs and finite vehicle fleets. PyVRP utilizes a genetic algorithm combined with local searching in Python, whereas Jsprit is implemented in Java and employs a ruin-and-recreate algorithm for solution search. We adapt these frameworks to our problem setup by scaling the instances by a factor 10,000 and formulating the problem as a prize-collecting VRP (PC-VRP) with TW for PyVRP, and as a CVRP with TWs and finite vehicle fleet for Jsprit. For the time constraints, we set the customer TW constraints from zero to the given customer end times e^n and run 10,000 iterations to ensure comparable results. The prize-collecting factor, which rewards the PyVRP agent for visiting customers, was calibrated for each instance. At low values, PyVRP minimizes distance but does not fully utilize vehicle capacity. By iteratively increasing the prize by 50, we identified the minimum value required to achieve full fleet utilization in PC-VRP. Calibration was performed before the solution search for each instance and is, therefore, not included in the PyVRP runtime. Table 2

shows the results, as solution quality, for Jsprit, PyVRP, and the trained model.

For validation, eight cases were generated, either with varying distributions (seeds) or different numbers of customers. The model was trained on random instances with 50 customers of the seed 1234, while being evaluated every 100 updates using seed 4321. This means that only the first two instances, that is, N50a and N50b, were known to the agent from the training, while all other instances have different numbers of customers. Instance N1000c has a new customer distribution (seed 3241), which was not contained in the training and evaluation datasets. In this way, we aim to test the transferability and generalization of our model. The solutions were analyzed with respect to the total distance traveled by the vehicle fleet (d), the number of customers served (N), fleet utilization (calculated as total fleet capacity minus total served demand) in percentage (FU), and the computation time (t) on the central processing unit (CPU) for the heuristics, as well as on both the CPU and the graphics processing unit (GPU) for NCO inference.

As shown in Table 2, the PAMP method achieves results that are comparable to or better than PyVRP in instances N50a, N50b, and N100a with respect to the total traveled distance. However, the NCO agent serves fewer customers than both PyVRP and Jsprit (e.g., 27 customers by 100% fleet utilization versus 87 customers served in Jsprit for N1000c). This indicates that the PAMP policy is strongly driven by demand collection, leading it to prioritize high-demand customers to maximize fleet utilization. In contrast, PyVRP and Jsprit prioritize the number of visited customers. This is because the PAMP agent is penalized for not using the full vehicle capacity. Thus, the PAMP agent pays more attention to the nodes with high demand, resulting in fewer customers, whereas the heuristics attempts to

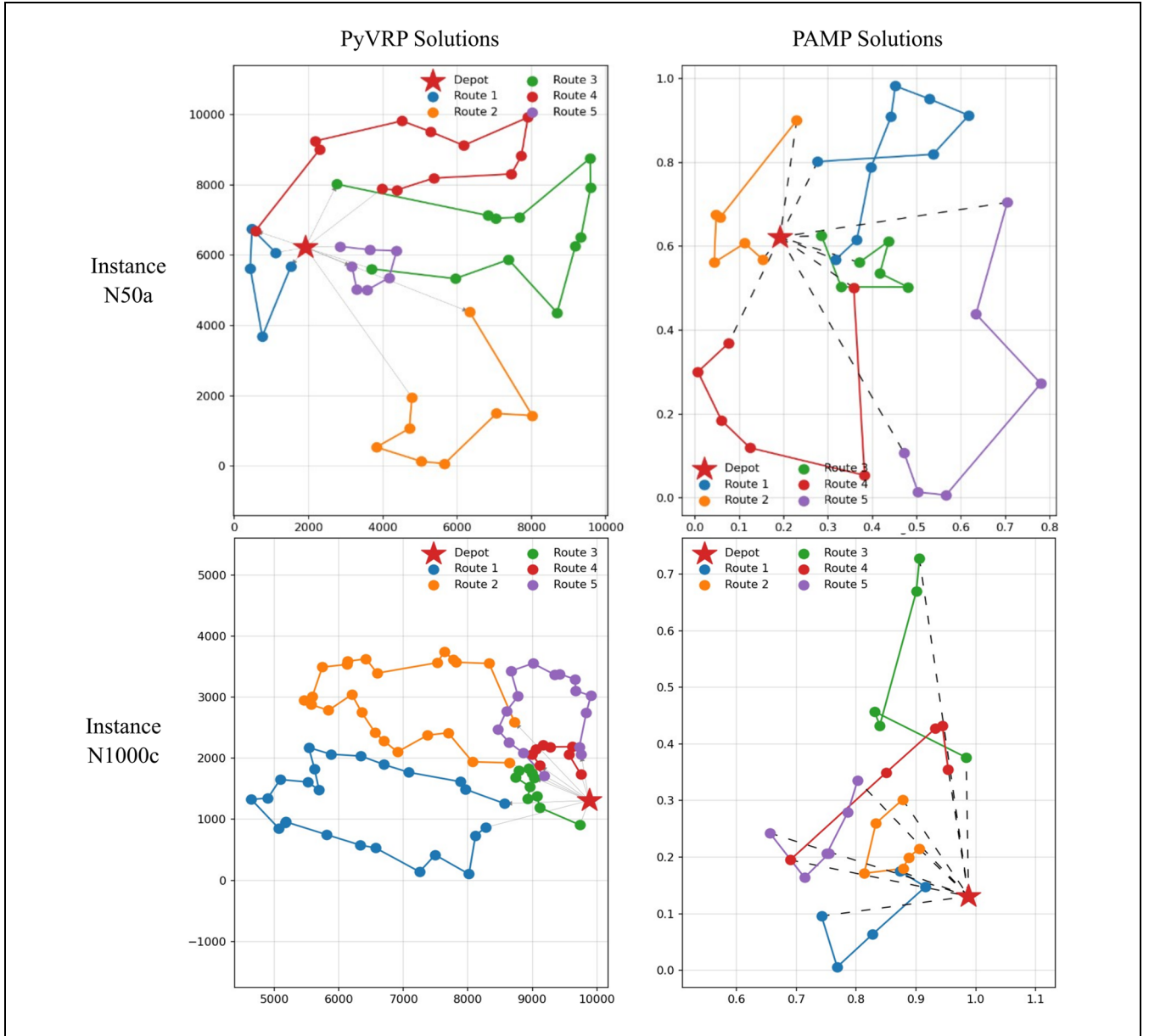


Figure 4. For instance 50a (top row) and 1000c (bottom row), operations research solutions (left column) and corresponding pointer network amplifier (PAMP) solutions (right column).

maximize the number of customers in a tour, resulting in longer tours. For small problems, where the number of possible optima is limited, the heuristics and NCO solutions are of the same magnitude (e.g., N50 and N100). However, the deviation becomes particularly evident as the number of possible alternative solutions grows with the size of the problem instance. In fact, we observed that, in oversized instances with more than 500 customers, the NCO agent prioritizes customers with high demand and tight TWs first, thereby utilizing the available fleet capacity more effectively while producing shorter tours. Furthermore, our method demonstrates

good transferability with respect to varying customer distributions, showing no notable variation across different seeds.

Figure 4 illustrates a comparison of routes generated by PyVRP and the PAMP method for two instances, N50a and N1000c. For instance N50a, the heuristics generated by PyVRP result in total costs of 7.42 distance units (top left), compared with the 6.66 distance units from the PAMP method (top right). In the large instance, N1000c, the PyVRP solution results in 3.71 distance units (bottom left) by 100% fleet utilization, while PAMP generates five tours with 4.53 distance units in

Table 3. PAMP ablation trained with default model: loose TW constraint distribution with TW: [10–500]

Index	Default				Mask-only				PAMP			
	<i>d</i>	<i>N</i> (%)	<i>D</i> (%)	ls	<i>d</i>	<i>N</i> (%)	<i>D</i> (%)	ls	<i>d</i>	<i>N</i> (%)	<i>D</i> (%)	ls
N50a	8.22	78.0	79.2	5	8.59	76.0	79.6	0	8.74	74.0	79.2	0
N100a	8.75	55.0	44.1	7	8.98	52.0	43.6	0	10.61	53.0	43.8	0
N500a	7.46	12.6	7.9	4	7.98	13.2	7.9	0	9.18	12.6	7.9	0

Note: ls = late services; PAMP = pointer network amplifier; TW = time window.

total (bottom right). In smaller instances, such as N50a, the routes constructed by the PAMP agent align in shape with those produced by the heuristic solutions. There are only slight route overlaps (e.g., Route 1 in PAMP solution; Figure 4, bottom right), probably because of time constraints. In all instances, the PAMP solutions result in fewer customers per tour, compared with the benchmark heuristics. This supports the hypothesis that the PAMP agent selects nodes with higher demand and tight end times. However, the overall route construction remains satisfactory. This outcome underscores the proposed approach's ability to generalize, even when applied to larger instances with and an unknown distribution.

As expected, the computation time required to obtain a solution (inference for NCO methods and combinatorial search for operations research (OR) heuristics) was approximately 4 times lower for PAMP in the worst case (8s CPU and 1.6s GPU for instance N1000a), and 21 times lower on GPU than for PyVRP. Training the PAMP model for 1,000,000,000 steps (or 8,000 global updates for 102,400 episodes each) demanded approximately 1 week on the CPU cluster machine (20 cores with 2.8 GHz). Both Jsprit and PyVRP were run on a desktop machine with an Intel i7-1355U CPU for 10,000 iterations each. We conducted the inference of the PAMP model on both desktop CPU and GPU architectures on Tesla T4 (see Table 2).

To isolate the effect of the PAMP mechanism, we conducted an additional experiment, comparing three configurations that differ in how the TW constraint is handled at inference. The experiment was designed to answer the following question: holding the trained policy fixed, which inference constraint mechanism performs better—a hard mask on the action distribution or a soft reweighting of the attention scores by PAMP? To provide a neutral reference for measuring the inference effect of PAMP, the base policy was retrained without any constraint handling. The hard TW mask was implemented in the environment for this series of experiments and was applied at the decoding step during the inference. The same base policy was then evaluated under three configurations: no intervention (default), the hard mask alone (mask-only), and the urgency scores injected in the

decoding (PAMP). This design ensures that the three configurations differ only in the intervention applied at inference. The base policy was trained for 5,000 global updates, which is fewer updates than in the earlier experiments. The absolute distance and service-rate values reported next are, therefore, higher than would be expected from a fully trained model. However, these absolute values are not the object of the analysis; the relevant comparison is the relative effect of each intervention applied to the same base policy.

All configurations were evaluated on identical test sets generated with the same random seeds used during training. We report four metrics: total tour distance *d*, the fraction of customers served (*N* %), the fraction of total demand served (*D* %), and the absolute count of late services (ls), defined as customer visits violating a TW. Three instance sizes were considered (*N* = 50, 100, 500). To assess the behavior of the policy under tight constraint conditions, we evaluated two regimes: a tight TW regime with TW drawn from [10, 100] (Table 3) and a looser regime with TW drawn from [10, 500] (Table 4).

Under the loose regime (Table 3), the default configuration produces a small number of late services in all three instances (5 at *N* = 50; 7 at *N* = 100; 4 at *N* = 500), confirming that the underlying policy cannot satisfy the constraint without intervention. Both mask-only and PAMP eliminate all violations across all instances. The associated distance cost is small for both interventions but consistently smaller for mask-only. Mask-only incurs a distance overhead of + 4.5% at *N* = 50 (8.59 versus 8.22), + 2.6% at *N* = 100 (8.98 versus 8.75), and + 6.9% at *N* = 500 (7.98 versus 7.47). PAMP incurs + 6.3% at *N* = 50 (8.74), + 21.3% at *N* = 100 (10.61), and + 23.0% at *N* = 500 (9.18). Demand-served values are comparable across the three models, while PAMP serves slightly fewer customers than the mask-only model (74% versus 76%). Within loose TW distribution, PAMP, therefore, achieves feasibility, with a moderate distance overhead and no degradation in demand satisfaction.

Under the tight regime (Table 4) the differences between the configurations become more pronounced. The default configuration produces many late services

Table 4. PAMP ablation trained with default model: tight TW constraint distribution with TW: [10–100]

Index	Default				Mask-only				PAMP			
	<i>d</i>	<i>N</i> (%)	<i>D</i> (%)	<i>ls</i>	<i>d</i>	<i>N</i> (%)	<i>D</i> (%)	<i>ls</i>	<i>d</i>	<i>N</i> (%)	<i>D</i> (%)	<i>ls</i>
N50a	8.22	78.0	79.2	17	11.80	58.0	72.0	0	22.59	54.0	58.0	2
N100a	8.74	55.0	44.1	28	15.18	40.0	37.7	0	19.97	35.0	34.1	4
N500a	7.46	12.6	7.9	29	4.60	9.2	7.9	0	12.23	8.8	7.1	3

Note: *ls* = late services; PAMP = pointer network amplifier; TW = time window.

(17 at $N = 50$, 28 at $N = 100$, 29 at $N = 500$), reflecting that the policy violates the constraint frequently when no intervention is applied. Mask-only eliminates all violations by construction. Its distance overhead is +43.5% at $N = 50$ (11.80 versus 8.22) and +73.6% at $N = 100$ (15.18 versus 8.75). At $N = 500$, the masked tour is shorter than the default tour (4.60 versus 7.47, −38.4%), which is consistent with the mask removing infeasible candidates that the default policy would otherwise have visited at the cost of long detours. PAMP reduces violations relative to the default but does not eliminate them (from 17 to 2 at $N = 50$, 28 to 4 at $N = 100$, and 29 to 3 at $N = 500$). Its distance overhead is the largest of the three configurations: +175% at $N = 50$ (22.59), +128% at $N = 100$ (19.97), and +63.8% at $N = 500$ (12.23). The demand-served fractions under PAMP are slightly lower for $N = 100$ and $N = 500$, but PAMP yields the minimum at $N = 50$ relative to the mask-only (58% versus 72%).

Overall, mask-only provides exact feasibility at a moderate distance cost in both regimes. PAMP, when applied only at inference, eliminates violations under the loose regime but not under the tight TW regime, and incurs a higher distance cost. Customers and demand served under PAMP are comparable to mask-only for medium and large instances in the loose regime, but lower in the tight regime and for small instances.

Discussion

The proposed problem formulation belongs to the class of multiobjective VRPs. This type of problem frequently arises in practice as a biobjective optimization problem that combines minimization of total travel time with the provision of priority-based service to time-critical customers. A representative example is the dynamic courier, express, and parcel segment, where high-priority, last-minute shipments must be served first, owing to the high value of time. During peak-demand periods, carriers may replan and retain some lower-priority parcels to consolidate loads and use their limited fleet more efficiently on the following day. We modeled this problem by introducing a penalty factor for unused vehicle capacity, which

encourages the trained agent to prioritize customers with high demand and extended the attention mechanism in the pointer network. Validating the solutions shows that the PAMP method systematically inserts fewer customers into the solution while fully utilizing the available vehicle fleet under the calibrated reward function at $P = 10$, whereas the heuristics tend to maximize the number of customers served. Imagine a situation where the value of time for each unit of demand is high and customers with high demand (or with other priority reasons) must be prioritized. In this case, our approach behaves correctly, even if fewer customers are served overall. PAMP aims to reduce the total fleet travel distance while maximizing vehicle utilization. In contrast, state-of-the-art heuristics tend to focus more on the number of customers served and assign penalties (or prizes) based on customer visits. Furthermore, we examine how the structural features of NCO enable solution construction by leveraging the complete state representation and global context. In contrast, OR heuristics rely on sequential combinatorial searching to build and evaluate candidate routes. This potentially enables NCO methods to integrate more contextual information, such as vehicle-specific constraints, heterogeneous fleet rules, or service requirements, such as pickup-delivery or mandatory customer services.

A key computational aspect is the difference between NCO inference and classical OR combinatorial searching. The two approaches differ fundamentally in their computational logic and are not directly comparable. Training deep learning models relies on parallel tensor multiplications optimized for GPUs, enabling fast, scalable inference with computational time growing near linearly with problem size (50). In contrast, OR solvers use combinatorial search methods to explore large neighborhoods, which often leads to exponential increases in computational time as problem complexity grows (51). To illustrate this distinction, we reported computation times for OR heuristics on different instance sizes and compared them with NCO inference times on both CPU and GPU architectures. The results confirm that NCO inference is faster and more scalable, particularly on GPU architectures, making it well-suited for real-time, large-scale applications and integrated optimization

tasks. In these contexts, the agent is not designed to replace OR heuristics but rather to complement them by providing rapid, approximate, solutions that can be incorporated in broader optimization frameworks supporting heuristic methods (52). For example, for location routing problems, vehicle counts can be selected and locations optimized by heuristics, after which pretrained policies provide rapid route approximations. The experiments also demonstrated that additional constraints can be easily integrated in an agent's policy by injecting them in attention score calculation networks, allowing for flexible adjustment to different problem requirements.

However, several limitations should be highlighted. First, adapting the model to different fleet configurations or new constraints (e.g., different fleet sizes or heterogeneous vehicle types) requires full retraining, complicating its practical implementation. Although some frameworks are designed with the aim of creating a general NCO model (47), the development of a universal policy that functions across several problem domains without retraining is a subject of ongoing research. The PAMP architecture contributes to this goal by providing flexible policy mechanisms. Second, inference time increases with instance size, and performance degrades when the number of available vehicles deviates from the training configuration. For example, a model trained on 5 vehicles yielded acceptable but suboptimal routes when applied to 10 vehicles, and failed completely when applied to 20, indicating an optimality gap outside the training distribution. Third, the penalty parameters significantly influence the solution structure. We calibrated and analyzed these, as described in the section on methodology, and provide insights into how the policy behaves under low and high penalty signals. The penalty value requires careful calibration, depending on the operational objective, as it directly governs the trade-off between distance minimization and vehicle fleet utilization. For new problem setups, these parameters need to be recalibrated to achieve a feasible balance between objectives.

Several observations follow from the constraint-violation experiments. First, PAMP does not eliminate violations under tight TW conditions. The results in Tables 3 and 4 indicate that a policy trained under loose TWs consistently violates constraints when applied to tight TWs. In safety-critical deployments, this is decisive in favor of the hard mask. Second, both the distance and the service-rate costs of PAMP are higher than those of mask-only, especially under tighter constraints. PAMP, as currently formulated, always prioritizes urgency over distance, which means that it makes detours to handle time-pressure that the mask-only baseline can route around more efficiently. While this urgency awareness can be feasible in environments with loose constraints, it becomes counterproductive under pressure, as the agent

ignores distance-optimal customers in favor of those closer to their deadlines. Ideally, the policy should be trained on a distribution that covers both the upper and lower bounds of the target instance constraints. While retraining policy under tight TWs in our experiments did not lead to measurable improvement, this aspect merits deeper analysis in future work.

Nevertheless, PAMP demonstrates properties that cannot be naturally realized by the mask mechanism. It does not require the constraint to be evaluated at each decoding step, which makes it applicable in settings where the mask is not. The most relevant cases are dynamic and stochastic routing problems in which TWs are revealed online or shift during execution, partially observable settings in which the agent does not have full information about future feasibility, and large-scale or real-time problems in which the per-step feasibility check is too expensive to evaluate exhaustively. PAMP is also more flexible; the urgency term is a continuous signal that can be modulated, weighted against other objectives, or extended to soft preferences, such as customer priority or service-level targets, none of which can be expressed by a binary mask.

A further restriction is scale dependence: models trained on normalized instances cannot be applied to arbitrary coordinates (e.g., Solomon benchmarks adapted for a finite fleet) without a prenormalization module to ensure consistent problem properties. However, to allow readers to position our results within the broader VRP literature, we additionally outline how standard CVRP TW benchmark families can be adapted to our framework. Customer coordinates should be normalized to the unit square and demands expressed as fractions of vehicle capacity, as detailed in the section on the PAMP method. Because our framework relies on a specific calibration of TW ratios, the benchmark TW must be additionally rescaled to match the training distribution. However, problem representation in our environment and benchmark instances, such as the Solomon benchmark, exhibit structural differences in the time representations and vehicle fleet utilization. In this case, the Solomon benchmark specifies a ready time, a due time, and a service time for each customer, while our environment determines a single deadline per customer, measured from the moment the vehicle leaves the depot, and the clock is reset to zero at every depot departure. The adaptation to the well-known benchmark instances will require a relaxation of the original Solomon problem in two respects: (i) ready times have to be discarded and (ii) the per-vehicle clock reset in our environment also means that each vehicle measures customer deadlines from its own depot departure, rather than from the global start of the working day, as in the Solomon benchmarks. Reported feasibility cannot therefore feasibly

compared with the Solomon best-known results for CVRP TW. For this reason, in this work, we do not include a Solomon evaluation. A direct comparison would require either modifying the environment to support the Solomon formulation, concerning the TWs and finite fleet, or retraining the model on these benchmarks. Both deliberate next steps were outside the scope of this study.

Overall, the problem addressed in this work is relevant to real-world operations, especially in dynamic environments, where decision-making must be fast and feasible. Our findings highlight the method's generalization ability and substantial reduction of computation time for generating solutions, indicating that it can be incorporated in hybrid optimization frameworks, for example, alongside traditional heuristic methods. The direct incorporation of problem constraints through the PAMP mechanism appears to be scalable to various practical routing problem formulations, such as prioritizing critical services, enforcing the use of charging stations for electric vehicles, or giving more attention to customers with specific service requirements.

Conclusion and Outlook

This paper presents an end-to-end NCO model for a capacitated VRP with time constraints and a finite vehicle fleet—an NP-hard, multiobjective combinatorial problem often encountered in the context of transport logistics. Here, we adopted the encoder-decoder policy, trained by the PPO algorithm (44). Therefore, we adjusted the POMO protocol based on Kwon et al. (7) following the implementation scheme proposed by Wan et al. (3), which is the most efficient and flexible method for training the end-to-end NCO algorithms. A PAMP approach was introduced to handle dynamic customer constraints, such as TWs. By rescaling attention scores based on direct environment information, we presented an easy-to-follow alternative for implementing dynamic constraints in sophisticated encoder-decoder architectures. Through the amplification of attention scores, our method prioritized customers with urgent services during the tour construction. It can be easily extended to other soft constraints, such as additional service fees, customer preferences, or distance constraints for electric vehicles. Moreover, we successfully designed a reward function that penalizes the agent for incomplete trips, achieving multiobjective optimization. The tested model demonstrated reasonable and reliable results, especially on instances with less restrictive constraints, and showed reasonable generalization when applied to new problem instances. After the in-depth analysis of the results presented in the section on evaluation and validation, we demonstrated that the solutions produced by our method are both adequate and still

optimal. PAMP yields coherent routing behavior and substantially reduces TW violations relative to the unconstrained baseline, while retaining the flexibility that motivates the soft formulation.

However, validating the optimality of solutions produced by NCO is a nontrivial task, particularly in multi-objective and dynamic environments where standard benchmarks may be insufficient. The experimental results reveal a fundamental divergence in how learned soft biases and hard masking handle TW constraints. While PAMP provides a flexible learned mechanism, it is a probabilistic method that is vulnerable in tight constraint conditions, where the hard masking approach avoids by design. A natural next step is to evaluate the method on the Solomon benchmark instances—the standard reference for VRPs with TWs, which would situate our results within the established literature. This will require several adaptations, specified within this paper. We identify this evaluation as the immediate continuation of our work. Despite this, NCO shows growing promise for advancing optimization in transportation systems.

Authors' Note

This study employed generative AI tools exclusively to enhance wording, correct grammatical errors, and assist in debugging and adapting the code utilized in the research. Different AI models, all employing the GPT-4 architecture, were used at different stages of manuscript and code development.

ORCID iD

Elija Deineko  <https://orcid.org/0000-0003-1398-9711>

Author Contributions

The authors confirm contribution to the paper as follows: study conception and design: Elija Deineko, Carina Kehrt; data collection: Elija Deineko; analysis and interpretation of results: Elija Deineko, Carina Kehrt; draft manuscript preparation: Elija Deineko. All authors reviewed the results and approved the final version of the manuscript.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was funded by the German Federal Ministry for Digital and Transport (BMDV) as part of the mFUND funding initiative “Hyper networks of German Logistics—Using the potentials of hyper networks in freight transport and logistics—HEGEL” (funding number, 19F2201A).

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Data Accessibility Statement

The datasets generated or analyzed during this study are available in the GitHub repository: https://github.com/panik772-arch/RLOR_PAMP/tree/main

References

- Heggen, H., Y. Molenbruch, A. Caris, and K. Braekers. Intermodal Container Routing: Integrating Long-Haul Routing and Local Drayage Decisions. *Sustainability*, Vol. 11, No. 6, 2019, p. 1634.
- Barua, L., B. Zou, and Y. Zhou. Machine Learning for International Freight Transportation Management: A Comprehensive Review. *Research in Transportation Business & Management*, Vol. 34, 2020, p. 100453.
- Wan, C. P., T. Li, and J. M. Wang. RLOR: A Flexible Framework of Deep Reinforcement Learning for Operation Research. *arXiv Preprint arXiv:2303.13117*, 2023.
- Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv Preprint arXiv:1606.01540*, 2016.
- Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv Preprint arXiv:1707.06347*, 2017.
- Huang, S., R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 Implementation Details of Proximal Policy Optimization. *The ICLR Blog Track*, 2023. <https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>
- Kwon, Y. D., J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. *Advances in Neural Information Processing Systems*, Vol. 33, 2020, pp. 21188–21198.
- Simao, H. P., J. Day, A. P. George, T. Gifford, J. Nienow, and W. B. Powell. An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application. *Transportation Science*, Vol. 43, No. 2, 2009, pp. 178–197.
- Novoa, C., and R. Storer. An Approximate Dynamic Programming Approach for the Vehicle Routing Problem with Stochastic Demands. *European Journal of Operational Research*, Vol. 196, No. 2, 2009, pp. 509–515. <https://doi.org/10.1016/j.ejor.2008.03.023>
- Powell, W. B. A Unified Framework for Stochastic Optimization. *European Journal of Operational Research*, Vol. 275, No. 3, 2019, pp. 795–821.
- Powell, W. B. Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions. Wiley, Hoboken, NJ, 2022.
- Stimpson, D., and R. Ganesan. A Reinforcement Learning Approach to Convoy Scheduling on a Contested Transportation Network. *Optimization Letters*, Vol. 9, 2015, pp. 1641–1657.
- Goodson, J. C., J. W. Ohlmann, and B. W. Thomas. Rollout Policies for Dynamic Solutions to the Multivehicle Routing Problem with Stochastic Demand and Duration Limits. *Operations Research*, Vol. 61, No. 1, 2013, pp. 138–154.
- Hottung, A., and K. Tierney. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem. *arXiv Preprint arXiv:1911.09539*, 2019.
- Lu, H., X. Zhang, and S. Yang. A Learning-Based Iterative Method for Solving Vehicle Routing Problems. *8th International Conference on Learning Representations, ICLR 2020*, Addis Ababa, Ethiopia, 26 April 2020 to 1 May 2020.
- Da Costa, P. R. D. O., J. Rhuggenaath, Y. Zhang, and A. Akcay. Learning 2-Opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning. *arXiv Preprint arXiv:2004.01608*, 2020.
- Vinyals, O., M. Fortunato, and N. Jaitly. Pointer Networks. In *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lees, M. Sugiyama, and R. Garnett, eds.), Curran Associates, Inc., Red Hook, NY, USA, 2015, Vol. 28.
- Bello, I., H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural Combinatorial Optimization with Reinforcement Learning. *arXiv Preprint arXiv:1611.09940*, 2016.
- Nazari, M., A. Oroojlooy, L. Snyder, and M. Takác. Reinforcement Learning for Solving the Vehicle Routing Problem. *Advances in Neural Information Processing Systems*, Vol. 31, 2018, pp. 9861–9871.
- Vaswani, A., N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is All you Need. *arXiv Preprint arXiv:1706.03762*, 2017.
- Deudon, M., P. Cournut, A. Lacoste, Y. Adulyasak, and L. M. Rousseau. Learning Heuristics for the TSP by Policy Gradient. *Proc., Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018*, Delft, The Netherlands, June 26–29, 2018, Proceedings 15. Springer International Publishing, Cham pp. 170–181.
- Kool, W., H. Van Hoof, and M. Welling. Attention, Learn to Solve Routing Problems! *arXiv Preprint arXiv:1803.08475*, 2018.
- Wang, Q., and C. Tang. Deep Reinforcement Learning for Transportation Network Combinatorial Optimization: A Survey. *Knowledge-Based Systems*, Vol. 233, 2021, p. 107526.
- Bogrybayeva, A., M. Meraliyev, T. Mustakhov, and B. Dauletbayev. Learning to Solve Vehicle Routing Problems: A Survey. *arXiv Preprint arXiv:2205.02453*, 2022.
- Jungwirth, A., M. Frey, and R. Kolisch. The Vehicle Routing Problem with Time Windows, Flexible Service Locations and Time-Dependent Location Capacity. Technical University of Munich, Munich, Germany, 2020.
- Homburger, J., and H. Gehring. Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows. *INFOR: Information Systems and Operational Research*, Vol. 37, No. 3, 1999, pp. 297–318.
- Bräysy, O., and M. Gendreau. Vehicle Routing Problem with Time Windows, Part II: Metaheuristics. *Transportation Science*, Vol. 39, No. 1, 2005, pp. 119–139.
- Gong, Y. J., J. Zhang, O. Liu, R. Z. Huang, H. S. H. Chung, and Y. H. Shi. Optimizing the Vehicle Routing Problem with Time Windows: A Discrete Particle Swarm Optimization Approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 2, 2011, pp. 254–267.

29. Kritzing, S., F. Tricoire, K. F. Doerner, R. F. Hartl, and T. Stützle. A Unified Framework for Routing Problems with a Fixed Fleet Size. *International Journal of Metaheuristics*, Vol. 6, No. 3, 2017, pp. 160–209.
30. Eshragh, A., R. Esmailbeigi, and R. Middleton. An Analytical Bound on the Fleet Size in Vehicle Routing Problems: A Dynamic Programming Approach. *Operations Research Letters*, Vol. 48, No. 3, 2020, pp. 350–355.
31. Ahmed, Z. H., and M. Yousefikhoshbakht. A Hybrid Algorithm for the Heterogeneous Fixed Fleet Open Vehicle Routing Problem with Time Windows. *Symmetry*, Vol. 15, No. 2, 2023, p. 486.
32. Krasowski, H., J. Thumm, M. Müller, L. Schäfer, X. Wang, and M. Althoff. Provably Safe Reinforcement Learning: Conceptual Analysis, Survey, and Benchmarking. *arXiv Preprint arXiv:2205.06750*, 2022.
33. Spieker, H. Constraint-Guided Reinforcement Learning: Augmenting the Agent-Environment-Interaction. *Proc., 2021 International Joint Conference on Neural Networks (IJCNN)*, Shenzhen, China, IEEE, New York, 2021, pp. 1–8.
34. Solozabal, R., J. Ceberio, and M. Takáč. Constrained Combinatorial Optimization with Reinforcement Learning. *arXiv Preprint arXiv:2006.11984*, 2020.
35. Falkner, J. K., and L. Schmidt-Thieme. Learning to Solve Vehicle Routing Problems with Time Windows Through Joint Attention. *arXiv Preprint arXiv:2006.09100*, 2020.
36. Li, K., T. Zhang, R. Wang, Y. Wang, Y. Han, and L. Wang. Deep Reinforcement Learning for Combinatorial Optimization: Covering Salesman Problems. *Covering Salesman Problem. IEEE Transactions on Cybernetics*, Vol. 52, No. 12, 2021, pp. 13142–13155.
37. Li, J., Y. Ma, R. Gao, Z. Cao, A. Lim, W. Song, and J. Zhang. Deep Reinforcement Learning for Solving the Heterogeneous Capacitated Vehicle Routing Problem. *IEEE Transactions on Cybernetics*, Vol. 52, No. 12, 2021, pp. 13572–13585.
38. Peng, B., J. Wang, and Z. Zhang. A Deep Reinforcement Learning Algorithm Using Dynamic Attention Model for Vehicle Routing Problems. *Proc., Artificial Intelligence Algorithms and Applications: 11th International Symposium, ISICA 2019, Guangzhou, China, November 16–17, 2019, Revised Selected Papers 11*. Springer, Singapore, pp. 636–650.
39. Wang, Y., Y. H. Jia, W. N. Chen, and Y. Mei. Distance-Aware Attention Reshaping: Enhance Generalization of Neural Solver for Large-Scale Vehicle Routing Problems. *arXiv Preprint arXiv:2401.06979*, 2024.
40. Chen, J., and J. Luo. Enhancing Vehicle Routing Solutions Through Attention-Based Deep Reinforcement Learning. *Proc., 2023 5th International Conference on Data-Driven Optimization of Complex Systems (DOCS)*, Tianjin, China, IEEE, New York, 2023, pp. 1–9.
41. Xin, L., W. Song, Z. Cao, and J. Zhang. Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, No. 13, 2021, pp. 12042–12049.
42. Rabecq, B., and R. Chevrier. A Deep Learning Attention Model to Solve the Vehicle Routing Problem and the Pick-Up and Delivery Problem with Time Windows. *arXiv Preprint arXiv:2212.10399*, 2022.
43. Fitzpatrick, J., D. Ajwani, and P. Carroll. A Scalable Learning Approach for the Capacitated Vehicle Routing Problem. *Computers & Operations Research*, Vol. 171, 2024, p. 106787.
44. Huang, S., R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. M. Araújo. CleanRL: High-Quality Single-File Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, Vol. 23, No. 274, 2022, pp. 1–18.
45. Liang, E., R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, and I. Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. *Proceedings of the 35th International Conference on Machine Learning, PMLR*, Vol. 80, 2018, pp. 3053–3062.
46. Tessler, C., D. J. Mankowitz, and S. Mannor. Reward Constrained Policy Optimization. *arXiv Preprint arXiv:1805.11074*, 2018.
47. Berto, F., C. Hua, J. Park, L. Luttmann, Y. Ma, F. Bu, and J. Wang, et al. RL4co: An Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. *Proc., 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Vol. 2, Toronto, ON, Canada, Association for Computing Machinery, New York, 2025, pp. 5278–5289.
48. Wouda, N. A., L. Lan, and W. Kool. PyVRP: A High-Performance VRP Solver Package. *INFORMS Journal on Computing*, Vol. 36, No. 4, 2024, pp. 943–955.
49. Schroeder, S., M. Zilske, G. Liedtke, and K. Nagel. Computational Framework for a Multi-Agent Simulation of Freight Transport Activities. In: *Proceedings of the Transportation Research Board 91st Annual Meeting*, Vol. 1, Washington, D.C., 22–26 January 2012, p. 23. Accessed 1 June 2026.
50. Desislavov, R., F. Martínez-Plumed, and J. Hernández-Orallo. Compute and Energy Consumption Trends in Deep Learning Inference. *arXiv Preprint arXiv:2109.05472*, 2021.
51. M. Gendreau, and J.-Y. Potvin (eds.). *Handbook of Metaheuristics*, 2nd ed. Springer, New York, NY, USA, 2010.
52. Deineko, E., P. Jungnickel, and C. Kehrt. Learning-Based Optimisation for Integrated Problems in Intermodal Freight Transport: Preliminaries, Strategies, and State of the Art. *Applied Sciences*, Vol. 14, No. 19, 2024, p. 8642. <https://doi.org/10.3390/app14198642>