Learning to solve the Skill Vehicle Routing Problem with Deep Reinforcement Learning

Nayeli Gast Zepeda, André Hottung, and Kevin Tierney

Bielefeld University, Bielefeld, Germany
{nayeli.gast, andre.hottung, kevin.tierney}@uni-bielefeld.de

Abstract. Neural combinatorial optimization has proven effective in solving various simple routing problems including the traveling salesperson problem and the vehicle routing problem (VRP). However, real-world routing scenarios are usually significantly more complex, often requiring sophisticated methods to find even a single feasible solution. In this work, we apply neural combinatorial optimization to the more challenging skill VRP, where routes must be constructed for technicians with diverse skill sets while adhering to customer time windows. Due to the limited number of available technicians, finding feasible solutions is usually very challenging. We evaluate several state-of-the-art learning-based approaches on the skill VRP and explore different reward shaping techniques to penalize infeasible solutions during training. Our findings show that while most approaches can effectively solve instances with 20 customers, all approaches struggle to reliably find feasible solutions for instances with 50 customers.

Keywords: Neural Combinatorial Optimization · Deep Reinforcement Learning · Routing Problems

1 Introduction

Deep neural networks (DNNs) can be used to solve a variety of optimization problems, with a particular focus on vehicle routing problems (VRPs) in recent literature. There has been great progress with regard to problem sizes considered, VRP variants studied, and ever-improving architectures and methods proposed [2, 13, 14, 10]. However, the literature has primarily focused on problems for which it is trivial to find feasible solutions. This stands in stark contrast to practical applications, where finding feasible solutions may be difficult due to restrictive real-world constraints.

In this work, we present a neural combinatorial optimization (NCO) approach for solving the skill VRP. The skill VRP involves a set of technicians with varying skill sets that must serve customers with varying skill requirements during pre-specified time windows. This combination of constraints is a challenging one for NCO approaches, as it can lead to infeasible solutions, which create challenges for the reinforcement learning (RL) mechanisms used in NCO methods. A straightforward way of handling infeasibility is with a penalty, i.e., infeasible

solutions are assigned a value proportional to how much the constraints are violated. This is called *reward shaping* in the RL literature. We investigate penalty functions for NCO methods on the skill VRP in this work to gain insights into whether penalties are sufficient for RL methods to learn effective policies for finding feasible solutions.

The contributions of this work can be summarized as follows:

1. We introduce a Markov decision process (MDP) formulation for the skill VRP, enabling us to tackle this problem with various NCO techniques.
2. We present an instance generator that generates instances guaranteed to have at least one feasible solution. This generator can easily be extended to generate instances for other routing problems.
3. We compare different reward shaping techniques to incentivize feasible solution generation.

The remainder of the paper is organized as follows: Section 2 reviews related work on the skill VRP and NCO approaches for routing problems. Section 3 defines the mathematical notation for the skill VRP used in this paper. Section 4 presents our MDP formulation of the skill VRP, introduces the different reward formulations, and describes the generator for feasible instances. Finally, Section 5 reports the results of our experiments.

2 Related Work

2.1 The Skill Vehicle Routing Problem

The skill VRP is a well-known resource-constrained routing and scheduling problem. In these types of problems customers have certain demands that can only be met by a subset of the available vehicles or service operators from the start [16]. In the skill VRP, specifically, technicians with certain sets of skills service customers who in turn have specific skill demands, as shown in Figure 1.

The first flow-based mathematical formulation for the skill VRP was introduced in [5]. The problem formulation assumes different skill levels S, where each customer with skill demand S_j must be serviced by any one technician t that has skill level $S_t \leq S_j$. In [4] the authors extend the model to consider skill types instead of skill levels. A technician t can service a customer node j if $S_j \subseteq S_t$, i.e., if the technician provides all the skills/service types required by the customer. The authors in [6] extend the problem further to include time window constraints for customers (morning vs. afternoon), a special device, precedence and synchronization constraints between services. Travel cost is technician-dependent, i.e., more skilled technicians will be more expensive. In this work we model the skill VRP based on [4], but including time windows and service durations.

2.2 Deep Reinforcement Learning for Routing Problems

Neural Combinatorial Optimization (NCO) approaches apply machine learning methods and other neural techniques to solve combinatorial optimization (CO)

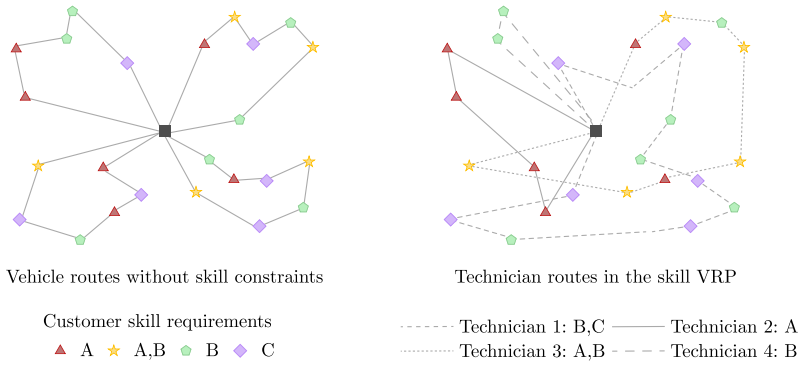

Fig. 1: Example of the skill VRP, based on [16, p.7].

problems. Specifically, NCO research has mostly applied to routing and scheduling problems, which are NP-hard, i.e., no polynomial-time algorithm is known to solve these problems optimally.

The foundation for NCO to routing problems was laid in [18], which introduces Pointer Networks (Ptr-Net). These networks can handle variable-size outputs that depend on the input size and produce a softmax probability distribution to effectively point to positions in the input sequence. The first application to routing problems was in [2], which uses Recurrent Neural Networks (RNNs) to encode city locations and sequentially construct Traveling Salesperson Problem (TSP) tours by predicting the next city to visit. [15] extends this approach to the Capacitated VRP (CVRP) by incorporating Reinforcement Learning (RL) to learn routing policies that respect vehicle capacity constraints while minimizing total route length.

Recent architectural improvements have further advanced the field. The AttentionModel (AM) [13], a transformer-based encoder with self-attention, processes all locations in parallel rather than sequentially, allowing the model to better capture global relationships between locations through multi-head attention mechanisms. This architecture also improves training stability and scalability to larger problem instances. Policy Optimization with Multiple Optima (POMO) [14] greatly increases performance while maintaining computational efficiency, by training with multiple starting locations and augmenting solutions through rotations and reflections during inference. The SymNCO method [12] extends this concept of leveraging problem and solution symmetries to enhance performance. Unlike POMO, SymNCO also incorporates augmentations during training to improve the model's generalization capability. In [20], the authors introduce MVMoE, a neural multi-task vehicle routing solver based on the mixture-of-experts (MoE) approach, designed to handle multiple VRP variants simultaneously. They report strong generalization in both zero-shot and few-shot settings. More recently, [11] propose PolyNet, a method that learns multiple diverse solution strategies using a single neural network. PolyNet demonstrates

strong performance, achieving near-optimal results, on the TSP with 100 nodes. All of these methods, however, have in common that they have only been evaluated on problems for which it is trivial to find feasible solutions. We go further and apply these methods to the skill VRP which has a challenging combination of constraints that can lead to infeasible solutions.

3 The Skill VRP: Notation

We build upon the skill VRP model introduced in [4]. Our model additionally includes service time windows as in [6] and service durations. Our goal is to provide a set of realistic constraints for investigating NCO methods.

Sets

T	Set of technicians.
N	Set of nodes (customers and depot).
N'	Set of nodes without the depot.
A	Set of feasible arcs between nodes $i, j \in N$ with $i \neq j$.
S_t^T	Set of skills offered by technician $t \in T$.
S_j^C	Set of skills required by customer $j \in N'$.

Parameters

c_t	Travel cost per distance unit for technician $t \in T$.
d_{ij}	Distance between $(i, j) \in A$, which we assume is equal to the travel duration.
s_j	Service time required at node $j \in N'$.
$[e_j, l_j]$	Time window for customer $j \in N'$, where e_j is the earliest start time and l_j is the latest.
H	System end time, which is the maximum route duration per technician.

Variables

$x_{tij} \in \{0, 1\}$	1 iff technician $t \in T$ travels on arc $(i, j) \in A$.
$z_j \in \mathbb{R}_0^+$	Arrival time of a technician at node $j \in N$.

Objective and Constraints

$$\min \sum_{t \in T} \sum_{(i,j) \in A} c_t d_{ij} x_{tij} \tag{1}$$

$$\sum_{(0,j) \in A} x_{t0j} \leq 1 \qquad \forall t \in T \tag{2}$$

$$\sum_{(i,j) \in A} x_{tij} = \sum_{(j,k) \in A} x_{tjk} \qquad \forall t \in T, j \in N \tag{3}$$

$$x_{tij} = 0 \qquad \forall (i,j) \in A, t \in T, S_j^C \not\subseteq S_t^T \tag{4}$$

$$\sum_{t \in T} \sum_{(i,j) \in A} x_{tij} = 1 \qquad \forall j \in N' \tag{5}$$

$$z_0 = 0 \tag{6}$$

$$e_j \leq z_j \leq l_j \qquad \forall j \in N' \tag{7}$$

$$z_i + s_i + d_{ij} \leq z_j + H(1 - \sum_{t \in T} x_{tij}) \qquad \forall (i,j) \in A, j \neq 0 \tag{8}$$

$$z_j + s_j + d_{j0} x_{tj0} \leq H \qquad \forall t \in T, (j,0) \in A \tag{9}$$

The objective minimizes the total travel cost for all technicians in Term (1). Constraints (2) indicate that every technician can leave the depot at most once. The flow balance Constraints (3) ensure that if a technician visits a customers, they must also leave the customer. Constraints (4) describe the skill constraints, i.e., if a technician t does not offer the skills S_j^C, they cannot visit customer j. Constraints (5) demand that every customer j be visited exactly once. The remaining constraints ensure temporal feasibility of the routes. The depot's start time is set in Constraint (6), Constraints (7) limit the arrival time to fall within the customer's time window. If a technician arrives at customer j before e_j, they have to wait for the time window to start, i.e., $z_j = e_j$. Constraints (8) ensure that when traveling from customer i to j, the arrival time z_j cannot be smaller than the sum of the arrival time at the previous customer plus the customer's service time and the travel distance. Finally, Constraints (9) ensure all technicians arrive back at the depot before the system end time H.

4 An MDP Formulation of the Skill VRP

Given a problem instance $\boldsymbol{x}$, the solution is generated through a **Markov Decision Process (MDP)** [1] defined as $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. The state space $\mathcal{S}$ describes the problem instance and the current partial solution at each step σ. The action space $\mathcal{A}_s$ includes all available actions at a given state $s \in \mathcal{S}$. Note that the action space can never be empty even if the current state corresponds to an infeasible solution. We describe how we model this in the context of the skill VRP in the following subsection. The state transition function $\mathcal{T}$ updates a state s_σ

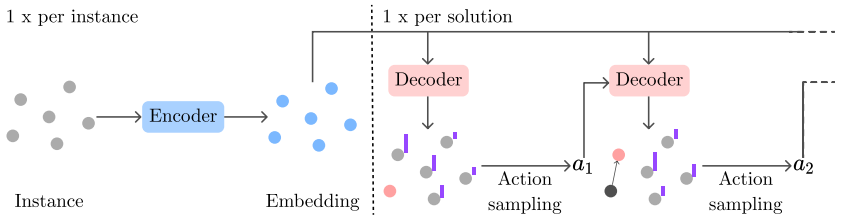

Fig. 2: Visualization of transformer-based RL, based on [9, 7]. The encoder maps the problem instance to an initial embedding that includes node locations, time windows, service durations, skill levels, and technicians' travel costs. Based on the current state $s \in \mathcal{S}$ which additionally includes the current time, node, and technician, as well as the partial solution, the decoder iteratively samples from the available actions $\mathcal{A}_s$ until all customers have been visited and the reward for the route can be calculated.

to the next state $s_{\sigma+1} = \mathcal{T}(s_\sigma, a_\sigma)$. The reward function $\mathcal{R}(s_\sigma, a_\sigma)$ represents the reward after taking action a_σ in state s_σ.

Routes are constructed by taking actions in the environment until all customer nodes have been visited, which is intuitively visualized in Figure 2. First, a trainable encoder f_θ maps the problem instance $\boldsymbol{x}$ to an embedding $\boldsymbol{h} = f_\theta(\boldsymbol{x})$. The decoder g_θ iteratively builds a solution by sampling from the available actions, based on the embedding $\boldsymbol{h}$ and the current partial solution, until all customer nodes have been visited. We call a sequence of T actions that defines a feasible route a *rollout* $\rho = (a_1, \ldots, a_T)$. The reward for the rollout is $R(\rho, \boldsymbol{x}) = \sum_{\sigma=1}^{T} \mathcal{R}(s_\sigma, a_\sigma)$. The encoding and decoding process is formalized as:

$$\pi_\theta(\rho|\boldsymbol{x}) \triangleq \prod_{\sigma=1}^{T-1} g_\theta(a_\sigma | a_{\sigma-1}, \ldots, a_0, \boldsymbol{h}), \tag{10}$$

where π_θ is the stochastic policy mapping the problem instance $\boldsymbol{x}$ to a rollout ρ.

We train the policy π_θ to maximize the expected reward $\mathbb{E}_{\rho\sim\pi_\theta(\rho|\boldsymbol{x})}[R(\rho, \boldsymbol{x})]$. For the distribution $P(\boldsymbol{x})$ of problem instances $\boldsymbol{x}$ the training objective becomes:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \left[\mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})} \left[\mathbb{E}_{\rho\sim\pi_\theta(\rho|\boldsymbol{x})} \left[R(\rho, \boldsymbol{x})\right]\right]\right]. \tag{11}$$

The REINFORCE algorithm [17] trains the solver π_θ by transforming eq. (11) into a minimization problem with a loss function, which can be optimized using gradient descent. The gradient for the REINFORCE loss function is given by:

$$\nabla_\theta \mathcal{L}(\theta|\boldsymbol{x}) = \mathbb{E}_{\pi(\rho|\boldsymbol{x})} \left[(R(\rho, \boldsymbol{x}) - b(\boldsymbol{x})) \nabla_\theta \log \pi(\rho|\boldsymbol{x})\right], \tag{12}$$

where $b(\cdot)$ is a baseline function that stabilizes training and reduces variance.

4.1 Specifying the Skill VRP

State Space $\mathcal{S}$ The state space includes both static information about the problem instance $\boldsymbol{x}$ and information about the partial solution, which is needed

to determine the available actions at each step. Static problem information includes the node locations in terms of their x and y coordinates, the skills of the technicians S_t^T, skill demands of the customers S_j^C, the customer time windows $[e_j, l_j]$ and service durations s_j, as well as the technician's travel cost c_t. Note that we use the Euclidean distance and consider a fully connected graph.

We encode the technicians each as separate depots, all identical except for the associated skill sets S_t and travel costs c_t. This way, we obtain a total of $|T|+|C|$ nodes, indexed from 0 to $|T| + |C| - 1$, where the first $|T|$ nodes correspond to technicians and the remaining $|C|$ nodes to customers. This allows us to encode the skills for each node j as a binary vector $s_j \in \{0, 1\}^K$, where K is the total number of distinct skills in the system. For technician nodes, $s_{jk} = 1$ indicates technician j offers skill k, and $s_{jk} = 0$ otherwise. For customer nodes, $s_{jk} = 1$ indicates customer j requires skill k, and $s_{jk} = 0$ otherwise. This binary vector representation allows for a straightforward extension to skill levels in the future by generalizing to integer-valued or continuous skill proficiency levels.

Information regarding the partial solution in the state space includes the set of visited nodes V (initialized as $V = \emptyset$), current node ν (initialized as $\nu = 0$), current time τ (determined by the travel durations and time windows of the nodes visited so far, initialized as $\tau = 0$), and the current technician t (chosen in the first step, determines the skills available and travel cost on the route). To facilitate learning, we also encode the remaining routes in the state, defined as $|T|$ minus the current number of routes in the solution. Every time a technician returns to the depot, a route is finished and added to the count. Note that any feasible solution will assign at most one route per customer. Finally, the current partial solution is given as the sequence of nodes visited so far in the rollout.

Action Space $\mathcal{A}$ The action space determines which nodes can be visited at each step. We use an action mask $\boldsymbol{a} \in \{0, 1\}^{|T|+|C|}$, where $\boldsymbol{a}_i = 1$ indicates node i can be selected at the current step and will be part of $\mathcal{A}_s$, and $\boldsymbol{a}_i = 0$ otherwise. In the first step the current technician t is set, i.e., only nodes with index $i < |T|$ are valid actions. To determine the subsequent available actions, we handle the mask separately for customer and depot nodes.

For customer nodes, we check four main conditions. First, we check for customers that have not been visited yet. Second, customers need to be reachable before their time windows end, considering the current time and travel distance to each node. Third, after servicing a customer, technicians need to be able to get back to the depot before the system end time H, i.e., we check for which nodes $\tau + s_j + d_{j0} \leq H$. Finally, only customers j can be visited for which the current technician t offers all required skills, i.e. $S_j^C \subseteq S_t^T$.

We only allow selecting a depot node if it has not been visited before, i.e., every technician should do at most one route, and if ν is a customer node or no more customers are left that can be serviced by t on the current route. This can lead to an empty action space $\mathcal{A}_s = \emptyset$ when all $|T|$ technicians have finished their routes but there are still unserved customers. To still allow training, we relax the condition that every technician can only do one route, i.e., for $\mathcal{A}_s = \emptyset$ all

technicians become available again. This will lead to a number of routes $> |T|$, but these infeasible solutions can be penalized in the reward function $\mathcal{R}$.

State Transition Function $\mathcal{T}$ We update the current node ν to the last sampled action. If the last action was to go back to a depot, we update the current technician t and increase the number of routes by 1. The current time is updated to $\tau_\sigma = \max(\tau_{\sigma-1} + d_{ij}, e_j) + s_j$ if ν is a customer node, i.e., s_j cannot start before e_j, and reset to $\tau_\sigma = 0$ at depot nodes.

Reward Function $\mathcal{R}$ We use a penalized objective function consisting of the sum of the route distances and a weighted penalty term, $\mathcal{R}(\rho, \boldsymbol{x}) = -(\lambda c_r + (1-\lambda)c_p)$. There are multiple options for how to compute c_p and how to set λ, which we describe below.

Penalty types We want to discourage the RL model from using more routes than available technicians. We use two simple penalty types for c_p. First, we penalize the number of excess routes E and set $c_p = |E|p_r$, where p_r is a penalty term set by the user. The reward function then is given as $\mathcal{R}^{Routes}(\rho, \boldsymbol{x}) = -(\lambda c_r + (1-\lambda)|E|p_r)$. Second, we penalize the route costs of all excess routes, setting $c_p = c_E p_r$, where c_E is the total route cost of excess routes. The reward is defined as $\mathcal{R}^{Cost}(\rho, \boldsymbol{x}) = -(\lambda a c_r + (1-\lambda)c_E p_r)$.

Reward weighing strategies The definition of the reward function allows for weighting the route cost and the penalty factor through the parameter λ. We define three strategies for setting the value of λ: (1) *fixed value*, (2) *batch-wise*, and (3) *exp. smoothed*. For *fixed value*, one constant value is used throughout training. For *batch-wise*, the average feasibility ratio is used, i.e., the ratio of feasible solutions in a training batch. In *exp. smoothed* we additionally smoothen the value from *batch-wise* with a smoothing factor of α. Both *batch-wise* and *exp. smoothed* depend on the feasibility ratio during training. The intuition behind these strategies is to lower the prominence of the penalty term once feasible solutions start to be found to avoid the search for feasibility from dominating the search for optimality.

4.2 Generating Feasible Instances

Much of the NCO literature on routing focuses on training models for problems where finding feasible solutions is trivial, but finding feasible solutions for the skill VRP is challenging. In particular, the number of vehicles is usually unlimited, however, in the skill VRP the number of technicians is inherently fixed. We posit that the model ought to be trained on feasible instances, as instances where there is no solution will always result in penalties. Thus, we need an instance generator that guarantees feasible instances.

The core idea is that instead of generating customer locations (x, y) and calculating distances $d_{i,j}$ based on the locations, we do the opposite. Each route

in a solution is constructed individually and sequentially such that the total travel duration including service times is always $\leq H$. The procedure is explained in Algorithm 1.

We first partition the $|N'|$ customers into $|T|$ routes in line 2. We simply allocate the customers randomly to routes, ensuring each route has between $[|N'|/|T| - |T|, |N'|/|T| + |T|]$ customers. The technicians are assigned skills on line 3. The number of skills per technician t, n_t^S, is determined by the user. For each technician t, we then build a route separately. For the number of customers $|M_t|$ on the route, we sample service times s in line 5 and subtract their total sum from H to get the time D that is actually available for traveling in line 6. We use the same unit for time and distance, so we split the remaining time D randomly into $C + 1$ travel legs d in line 7. We then sample the time windows $[e, l]$ in line 8 and customer skills S^C in line 9. Note that the customer time windows are only generated once d and s are available to ensure feasible time windows for each customer.

To generate the coordinates in line 10, we also need travel times d and service durations s. We construct the route sequentially by starting in the depot and iteratively sampling an angle θ out of all available angles such that (1) we do not leave the defined domain for (x, y) and (2) do not get so far away from the depot that the route cannot return to the depot before time H. Based on the angle θ_ι and travel leg d_ι at each step ι we update the current location before sampling the next angle, until we return to the depot and start the next route in the instance.

With this procedure we ensure that instances are random and potentially inefficient, while having at least one feasible solution. Note that the generation process is generic, except for the skill definitions, and can therefore easily be extended to other problems.

Algorithm 1 Feasible Instance Generation

1: **procedure** GenerateInstance($T, |N'|, S, c, H, n^S$)
2: $M \leftarrow$ PartitionCustomers($|T|, |N'|$)
3: $S_t^T \leftarrow$ a random subset of S of size n^S $\forall t \in T$
4: **for** t in T **do**
5: $s \leftarrow$ SampleServiceTimes(M_t)
6: $D \leftarrow H - \sum_{i \in M_t} s_i$ ▷ Available time for travel
7: $d \leftarrow$ SampleTravelLegs(M_t, D) ▷ Distances between customers
8: $e, l \leftarrow$ SampleTimeWindows(M_t, d, s)
9: $S^C \leftarrow$ SampleCustomerSkills(S, M_t)
10: $x, y \leftarrow$ GenerateCoordinates(d, s)
return Instance($S^T, S^C, s, d, e, l, x, y$)

5 Experiments

We compare five NCO approaches with regard to their ability to find feasible, high-quality solutions on the skill VRP and consider the following research questions:

- **RQ1** Can neural solvers effectively find feasible solutions for skill VRP instances as the problem constraints become more restrictive?
- **RQ2** Does training on datasets with only feasible instances significantly improve model performance compared to training on data that includes infeasible instances?
- **RQ3** How should the reward function be designed to effectively incentivize the construction of feasible solutions?

In all experiments, models are trained with the same seed on NVIDIA A100 GPUs, inference is performed on an Nvidia GeForce RTX 4090. We use the open-source RL4CO framework [3], which contains implementations of many of the latest state-of-the-art methods and allows us to build on current research.

5.1 Experimental Setup

Instance Generation Using the procedure described in Section 4.2, we generate two datasets, one with $n = 20$ customers and another with $n = 50$ customers. For both sets we set the system end time $H = 480$ and assume 3 technicians. One technician possesses all six skills and has travel cost $c_t = 2.0$, and two technicians offer four services with travel cost $c_t = 1.0$. The depot is located at the center of the instance, which is 100 by 100 units in size.

For instances with $n = 20$ customers, each customer requires three skills with probability 0.3 or one skill with probability 0.7. The service durations s_j follow a discrete distribution, where $P(s_j = 10) = 0.5$, $P(s_j = 20) = 0.3$, and $P(s_j = 30) = 0.2$. The time windows $[e_j, l_j]$ are assigned from the set $\{[0, 240], [240, 480]\}$. Each customer is assigned a time window with probability 0.5, otherwise they are assigned $[0, H]$.

For $n = 50$ customers, each customer requires three skills with probability 0.7 and one skill with probability 0.3. Service durations s_j follow the discrete distribution, where $P(s_j = 5) = 0.5$, $P(s_j = 10) = 0.35$, and $P(s_j = 20) = 0.15$. Time windows $[e_j, l_j]$ are assigned from the set $\{(0, 120), (120, 240), (240, 360), (360, 480)\}$. Each customer is assigned a time window with probability 0.8, otherwise they are assigned $[0, H]$.

Per dataset we generate 100,000 instances for training, 1,000 for validation, and 1,000 for testing. The models for $n = 20$ are trained for 100 epochs, and for $n = 50$ for 500 epochs. All instances are scaled down such that their locations lie in $[0, 1]$, time windows and service durations are also scaled down accordingly.

Models The models we use are AM [13], POMO [14], MVMoE-POMO [20], PolyNet [11], and SymNCO [12]. We set the batch size for training, validation

and test to 128 and the learning rate to 0.0001. For all models we use the default parameter settings as defined in the RL4CO framework.

During training, the AM approach uses a greedy rollout as baseline in eq. (12) and SymNCO uses its own baseline (see [12] for details), the other models use a shared baseline, i.e., the mean reward in the training batch is used as the baseline $b(\cdot)$ for the loss calculation.

At test time, all models except AM use instance augmentation. POMO, MVMoE-POMO and PolyNet use dihedral augmentation, i.e., instances are rotated and flipped in the domain, to achieve 8 different representations of the same instance [14, Table 1]. SymNCO uses symmetric augmentation, i.e., it randomly samples 10 rotation angles $\phi \in (0, 4\pi)$ and rotates the instances accordingly. All models except PolyNet perform greedy rollouts during inference. The AM approach generates a single solution per instance, while POMO and MVMoE-POMO generate $8 \times |T|$ solutions, SymNCO generates $10 \times |T|$, and PolyNet produces 8×800 solutions per instance.

Baselines As baselines we use version Gurobi 11.0.3 [8] and PyVRP 0.11.0a0 [19]. For Gurobi we set a per-instance time limit of 5 hours and for PyVRP 10 seconds. Note that PyVRP uses integer values in its objective function, thus its values are not as exact as for Gurobi. All gaps are computed as the gap to the value found by Gurobi, which is not always the optimal value due to the timeout.

Evaluation Metrics Our evaluation considers a total of four metrics. The feasibility *count* tells us for how many of the 1,000 instances the neural solvers find feasible solutions. The *cost* is the average route cost over the best feasible solution found for each instance (if any feasible solution was found). For the *gap %* calculation we take the percentage deviation of *cost* from the solutions provided by Gurobi. Lastly, we provide the runtime *time (s)* during inference.

5.2 RQ1: Feasibility Under Tight Constraints

We evaluate five neural solvers on their ability to find feasible solutions for skill VRP instances as the problem constraints become more restrictive, comparing their performance on $n = 20$ and $n = 50$ in Table 4. Note that the $n = 50$ instances also have tighter time windows and higher skill requirements. For an intuition on how difficult it is to find feasible solutions for these instances, we include the training curves reporting the average feasibility rates for all five neural solvers while training with $\mathcal{R}^{Routes}$ on $n = 20$ and $n = 50$ in Figure 3.

Except for AM, all models find feasible solutions for all $n = 20$ and most for $n = 50$ instances. That is, they find feasible solutions even as the constraints become more restrictive, but the rate decreases. PolyNet in particular stands out for its low gaps across experiments, while for $n = 50$ it finds the most feasible solutions for $\mathcal{R}^{Cost}$ and the second most for $\mathcal{R}^{Routes}$. As PolyNet produces more solutions per instance than any of the other models, this indicates that solution diversity may be crucial for finding high-quality, feasible solutions to constrained

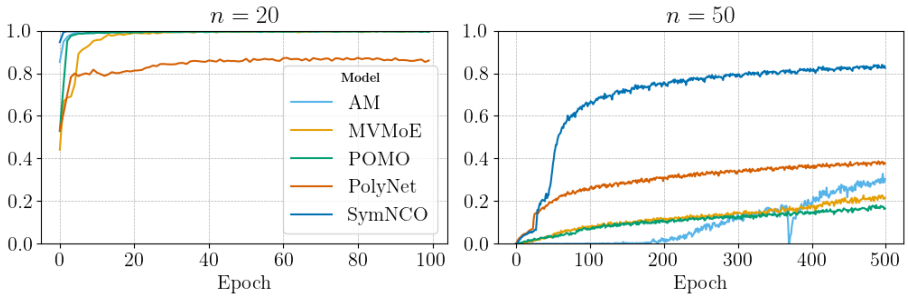

Fig. 3: Average feasibility rates during training using $\mathcal{R}^{Routes}$.

problems as the skill VRP. This becomes particularly apparent when considering Figure 3, where PolyNet does not have the highest average feasibility rates by any means. However, for the quality of the final solution during inference, we do not need all rollouts to provide feasible solutions. Instead, out of the feasible solutions provided by the model we can choose the one with the lowest cost.

Table 4: Performance of NCO models in finding feasible solutions for the skill VRP.

	$\mathcal{R}^{Routes}$							
	$n = 20$				$n = 50$			
	count	cost	gap %	time (s)	count	cost	gap %	time (s)
Gurobi	1000	642.2724	-	106.249	1000	986.9012	-	12398.9071
PyVRP	1000	642.4042	0.0002	10.0005	1000	984.0765	-0.0028	10.0009
AM	999	862.813	37.0574	0.045	406	1222.357	24.7333	0.097
POMO	1000	743.543	15.8578	0.146	903	1304.398	33.4175	0.289
MVMoE-POMO	1000	753.309	17.3870	0.217	936	1248.154	27.3745	0.378
PolyNet	1000	677.851	5.5058	0.158	956	1059.247	7.6277	0.477
SymNCO	1000	803.378	26.3096	0.080	988	1302.226	32.4832	0.210
	$\mathcal{R}^{Cost}$							
	$n = 20$				$n = 50$			
AM	1000	863.730	37.2030	0.044	377	1389.097	41.7479	0.099
POMO	1000	777.010	21.3393	0.147	947	1110.252	13.1906	0.290
MVMoE-POMO	1000	758.203	18.1570	0.187	836	1261.512	29.5602	0.378
PolyNet	1000	677.559	5.4488	0.154	965	1047.606	6.4838	0.313
SymNCO	1000	810.345	27.3681	0.080	953	1328.623	35.4243	0.211

5.3 RQ2: Training on Feasible vs. Purely Random Instances

To investigate the importance of feasible instances for training, we perform a set of experiments on instances generated such that feasibility is not guaranteed,

reporting the results in Table 5. We use the reward function $\mathcal{R}^{Routes}$ and apply the same models and training parameters as in Section 5.2.

In these instances, locations are generated randomly in the domain without enforcing a total travel time $\leq H$. With increasing node count in the domain we expect more instances to not have a feasible solution, simply due to increased travel duration while H and the number of technicians stay constant. We solve subsets of these instances in PyVRP to get a rough approximation about the feasibility rates in the instances. For $n = 20$ about 94% of instances were shown to be feasible within 10 seconds, for $n = 50$ none could be. This stands in contrast to our dataset of feasible instances, where PyVRP has no problem finding feasible instances. We note that we have not proven the instances to be infeasible with Gurobi due to the high computational expense of doing so for a large training set. For validation and testing we use data generated with algorithm 1.

We find that for $n = 20$ instances all models find feasible solutions for all instances. POMO even finds better solutions than when trained on feasible instances, for all other models the opposite is true. On the $n = 50$ instances the number of feasible solutions found is much lower than when trained on feasible instances. The difference is particularly large for SymNCO and PolyNet, where less than 50 feasible solutions are found, compared to over 950 instances when trained on feasible instances. Even considering that the models are evaluated out-of-distribution due to the different generation of node locations, this indicates the importance of feasible instances for effective training of neural solvers.

5.4 RQ3: Impact of Different Penalty Strategies

All experiments in this section are performed on SymNCO with identical settings, only varying individual parameters to examine their respective impact on solution quality during inference.

Fixed versus dynamic $\boldsymbol{\lambda}$ In Section 4.1 we introduce three simple strategies to determine the penalization factor λ. We compare the three strategies with

Table 5: Performance of models trained on randomly generated instances not guaranteed to be feasible. Experiments marked with * did not finish training for 500 epochs.

	$\mathcal{R}^{Routes}$							
	$n = 20$				$n = 50$			
	count	cost	gap %	time (s)	count	cost	gap %	time (s)
Gurobi	1000	642.2724	-	106.249	1000	986.9012	-	12398.9071
PyVRP	1000	642.4042	0.0002	10.0005	1000	984.0765	-0.0028	10.0009
AM	1000	890.339	41.4298	0.042	89	1418.150	44.7126	0.114
POMO	1000	764.681	19.5019	0.147	245	1385.925	47.5690	0.294
MVMoE-POMO	1000	771.810	20.3495	0.180	205	1399.120	49.1941	0.379
PolyNet	1000	682.425	6.3677	0.154	*46	1168.896	24.9038	0.305
SymNCO	1000	831.527	30.7165	0.079	48	1441.486	55.6091	0.203

Table 6: Different strategies for setting λ.

n	λ strategy	$\mathcal{R}^{Routes}$			$\mathcal{R}^{Cost}$		
		count	cost	gap %	count	cost	gap %
20	fixed 0.1	1000	808.563	27.1285	1000	807.052	26.8868
	fixed 0.2	1000	799.454	25.6306	1000	811.005	27.4494
	fixed 0.5	1000	807.909	27.0326	1000	809.790	27.2659
	batch-wise	1000	703.591	9.4615	1000	755.943	17.8799
	exp. smoothed	1000	785.667	23.4467	998	755.940	17.8337
50	fixed 0.1	990	1302.966	32.4852	958	1325.401	35.0077
	fixed 0.2	993	1305.460	32.7640	976	1317.151	34.1511
	fixed 0.5	988	1302.226	32.4832	953	1328.623	35.4243
	batch-wise	990	1306.875	32.9245	983	1314.946	33.8060
	exp. smoothed	992	1300.193	32.1998	975	1318.165	34.2188

regard to their performance on $n = 20$ and $n = 50$ in Table 6. For fixed λ we further evaluate different values. For exponentially smoothed λ we set $\alpha = 0.01$.

For $n = 20$, the exponentially smoothed λ is the only strategy that does not find feasible solutions for all 1,000 instances when applying $\mathcal{R}^{Cost}$. For $\mathcal{R}^{Routes}$, all strategies find feasible solutions for all instances, but batch-wise λ finds solutions with the lowest costs. For $n = 50$, exponentially smoothed λ and fixed $\lambda = 0.2$ find the most solutions for $\mathcal{R}^{Routes}$, but all strategies have similar results regarding the feasibility count and route cost. For $\mathcal{R}^{Cost}$ differences are more apparent and the batch-wise strategy seems to out-compete fixed and exponentially smoothed, finding the most feasible solutions on $n = 50$ while maintaining the lowest costs on both $n = 20$ and $n = 50$.

We further see that $\lambda = 0.1$ may not always be the best choice when opting for a fixed value. With regard to the number of feasible solutions found, $\lambda = 0.2$ is the the best of the values tested, as the model finds more feasible solutions for $n = 50$, both for $\mathcal{R}^{Routes}$ and $\mathcal{R}^{Cost}$. With regard to route cost on feasible solutions, it finds the lowest costs for $n = 50$ with $\mathcal{R}^{Routes}$, while for $\mathcal{R}^{Cost}$ the cost is actually the highest. For $n = 50$ this effect is reversed, but in any case the route costs are relatively close. Choosing $\mathcal{R}^{Routes}$ for the reward function seems a dominant strategy, as these models consistently find more feasible solutions, particularly for $n = 50$.

Effect of Penalty Size We investigate the effect of higher values for the penalty term p_r. We hypothesize that higher values ought to lead to more feasible solutions, but at the expense of finding solutions with low costs. Table 7 provides our results. We find that for lower p_r values, SymNCO is able to consistently find solutions with lower cost when using $\mathcal{R}^{Routes}$ for the reward, which suggests using lower values might be preferable. Note that instances are scaled down to lie in $[0, 1]$ for training, therefore $p_r = 10$ is still a meaningful penalty size, depending on the route cost. For this reason, for $n = 50$, where routes are longer, we see that $p_r = 10$ comes at the expense of finding fewer feasible solutions. It depends on the difficulty and size of the problem which penalty size is preferred.

For $\mathcal{R}^{Cost}$ we see the same effect for $n = 20$, i.e., lower penalties resulting in lower costs. For $n = 50$, on the other hand, $p_r = 100$ seems to strike a good balance between penalizing infeasibility and minimizing route costs.

Table 7: The impact of different penalty sizes.

n	p_r	$\mathcal{R}^{Routes}$			$\mathcal{R}^{Cost}$		
		count	cost	gap %	count	cost	gap %
20	10	1000	781.389	22.7755	1000	779.776	22.5761
	100	1000	795.328	25.0404	1000	801.331	25.9821
	1000	1000	807.909	27.0326	1000	810.345	27.3681
50	10	964	1281.325	30.5171	824	1353.882	38.9164
	100	988	1300.021	32.2313	986	1308.405	33.0451
	1000	988	1302.226	32.4832	953	1328.623	35.4243

Alternative Reward Functions Our results show that neither reward function dominates the other across all problem sizes and models. In our models trained with SymNCO (Tables 6 and 7), $\mathcal{R}^{Routes}$outperforms consistently. Most models also perform best with $\mathcal{R}^{Routes}$. However, POMO finds better solutions using the $\mathcal{R}^{Cost}$function on $n = 50$, see Table 4. The main conclusion is that the reward function plays a significant role in the quality of the resulting model.

6 Conclusion

In this work, we applied NCO approaches to the challenging skill VRP. We formulated the solution generation process as a Markov Decision Process, incorporating penalty mechanisms for infeasible solutions, and evaluated various learning-based optimization methods from the literature. To facilitate effective training, we designed an instance generator that ensures at least one feasible solution per problem instance. Furthermore, we explored different penalization and reward strategies to guide the models toward feasible solutions. Our results demonstrate that NCO approaches can effectively tackle complex routing problems, opening avenues for future research on more constrained and real-world VRP scenarios.

Acknowledgments. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - No. 521243122. The authors gratefully acknowledge the computing time provided to them on the high-performance computer Noctua 2 at the NHR Center PC2. These are funded by the Federal Ministry of Education and Research and the state governments participating on the basis of the resolutions of the GWK for the national high-performance computing at universities (www.nhr-verein.de/unsere-partner).

Disclosure of Interests. The authors have no competing interests.

References

1. Bellman, R.: A markovian decision process. Indiana University Mathematics Journal **6**, 679–684 (1957)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural Combinatorial Optimization with Reinforcement Learning. No. arXiv:1611.09940, arXiv (2017)
3. Berto, F., Hua, C., Park, J., Luttmann, L., Ma, Y., Bu, F., Wang, J., Ye, H., Kim, M., Choi, S., Zepeda, N.G., Hottung, A., Zhou, J., Bi, J., Hu, Y., Liu, F., Kim, H., Son, J., Kim, H., Angioni, D., Kool, W., Cao, Z., Zhang, Q., Kim, J., Zhang, J., Shin, K., Wu, C., Ahn, S., Song, G., Kwon, C., Tierney, K., Xie, L., Park, J.: RL4CO: an extensive reinforcement learning for combinatorial optimization benchmark. In: 31st SIGKDD Conference on Knowledge Discovery and Data Mining - Datasets and Benchmarks Track (2025)
4. Cappanera, P., Gouveia, L., Scutellà, M.G.: Models and valid inequalities to asymmetric skill-based routing problems. EURO Journal on Transportation and Logistics **2**(1), 29–55 (May 2013)
5. Cappanera, P., Gouveia, L., Scutellà, M.G.: The skill vehicle routing problem. In: Pahl, J., Reiners, T., Voß, S. (eds.) Network Optimization. pp. 354–364. Lecture Notes in Computer Science Volume 6701, Springer
6. Cappanera, P., Requejo, C., Scutellà, M.G.: Temporal constraints and device management for the skill vrp: mathematical model and lower bounding techniques. Computers & Operations Research **124**, 105054 (2020)
7. Gast Zepeda, N., Hottung, A., Tierney, K.: Deep learning in search heuristics. In: Martí, R., Pardalos, P.M., Resende, M.G. (eds.) Handbook of Heuristics. pp. 1–18. Springer Nature Switzerland, Cham (2025)
8. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024)
9. Hottung, A., Kwon, Y.D., Tierney, K.: Efficient active search for combinatorial optimization problems. In: International Conference on Learning Representations (2022)
10. Hottung, A., Wong-Chung, P., Tierney, K.: Neural deconstruction search for vehicle routing problems. Transactions on Machine Learning Research (2025)
11. Hottung, A., Mahajan, M., Tierney, K.: PolyNet: Learning diverse solution strategies for neural combinatorial optimization. In: International Conference on Learning Representations (2025)
12. Kim, M., Park, J., Park, J.: Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 1936–1949. Curran Associates, Inc. (2022)
13. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019)
14. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In: Advances in Neural Information Processing Systems. vol. 33, pp. 21188–21198. Curran Associates, Inc. (2020)
15. Nazari, M., Oroojlooy, A., Snyder, L., Takac, M.: Reinforcement learning for solving the vehicle routing problem. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
16. Paraskevopoulos, D.C., Laporte, G., Repoussis, P.P., Tarantilis, C.D.: Resource constrained routing and scheduling: Review and research prospects **263**(3), 737–754

17. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems **12** (1999)
18. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc. (2015)
19. Wouda, N.A., Lan, L., Kool, W.: PyVRP: a high-performance VRP solver package. INFORMS Journal on Computing **36**(4), 943–955 (2024)
20. Zhou, J., Cao, Z., Wu, Y., Song, W., Ma, Y., Zhang, J., Chi, X.: MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts. In: Proceedings of the 41st International Conference on Machine Learning. pp. 61804–61824. PMLR (2024)