Proceedings of the 10th Annual World Conference of the
Society for Industrial and Systems Engineering,
2021 SISE Virtual Conference
September 23-24, 2021

A Two-Stage Deep Reinforcement Learning Framework for Solving Large Capacitated Vehicle Routing Problems

A. Arishi, K. Krishnan, and V. Maru

Department of Industrial, Systems and Manufacturing Engineering,
Wichita State University
Wichita, KS 67220, USA

Corresponding author's Email: aaarishi@shockers.wichita.edu

Abstract: The Vehicle Routing Problem (VRP) is one most studied NP-hard combinatorial optimization problems due to its practical importance and methodological interest. Yet despite considerable progress, existing approaches including traditional heuristics and learning methods, struggle to find a fast and high-quality solution for large-scale VRPs. Most of the recent works using Deep Reinforcement Learning (DRL) for VRPs ignore the scalability aspects as they solve small instances with at most 100 customers. Also, they do not consider real-world or complex settings as they only test the trained model on random datasets generated from the same distribution with similar characteristics. This paper proposes a two-stage DRL framework to approximately solve the large capacitated vehicle routing problems (CVRP). In this framework, a DRL with an attention mechanism is trained using early stopping to increase generalizability and avoid overfitting on small CVRPs. Then, the trained DRL is combined with a local improvement 2-opt heuristics to further improve the solution. Moreover, as opposed to other works, the proposed approach is tested on several classical benchmark instances to investigate the scalability and generalizability. These instances are generated to fill the gap between the real world and simulation and pose a significant challenge even for modern heuristics and metaheuristics. The results show that a DRL trained on small-scale CVRP100 generalizes and scales well to larger instances. More importantly, hybridization using the 2-opt algorithm improves the solution quality and overcomes the drawback of the learned heuristics. When tested on different benchmark problems, the proposed framework outperforms Google's optimization tools (OR-tools) and classical heuristics in the aspect of both solution quality and execution time. This approach is desirable for practical CVRPs that are usually complex, large-scale and requires real-time decisions.

Keywords: Deep Reinforcement Learning, Large Capacitated Vehicle Routing Problem, Hybrid Model

1. Introduction

As VRP is a hard combinatorial problem, sophisticated exact algorithms such as column generation, branch-and-bound, branch-and-cut, branch-and-price can find an optimal solution only for small problems in a reasonable time. Thus, using them to solve large-scale VRP problems becomes impractical when response time is crucial. Therefore, researchers have focused on developing intuitive methods for solving these challenging problems. Several constructive heuristics, local improvement heuristics, and metaheuristics algorithms are introduced in literature; they can provide a suboptimal solution within an acceptable time for large-scale problems (Vidal, Crainic, Gendreau, & Prins, 2013). Constructive heuristics attempt to produce solutions constructively in a greedy manner using handcrafted rules. Examples of these heuristics include Clarke-Wright Savings, Cheapest Insertion and Sweep algorithms. However, handcrafted heuristics tend to be problem-specific, and hence designing them for combinatorial optimization becomes a non-trivial task. These classical heuristics are relatively simple to implement but suffer from a limited search space as they stop at the local optimum. Metaheuristic methods are designed to escape the local optima through a wider exploration of search space. These Metaheuristic methods have become a practical way of solving VRP and its variants. Popular metaheuristics used for solving VRP problems include Ant Colony Optimization (AOC), Simulated Annealing (SA), Tabu Search (TS), iterated local search (ILS), and Genetic Algorithm (GA). Despite their popularity, the implementation of metaheuristics requires many design choices and parameters tunning for the algorithm to perform efficiently. Thus, the quality of metaheuristics algorithms can vary greatly depending on the values of these parameters and the selected random seed, which makes comparing them a challenging task (Labadie, Prins, & Prodhon, 2016). In modern operation research, constructive heuristics are used to generate initial solutions for metaheuristics. Therefore, a weak initial solution can increase the search space, affecting the solution time and quality of metaheuristics algorithms. Local improvement heuristics works by taking an existing final solution delivered by construction or metaheuristics as an initial solution. Given

the current initial feasible routing, iteratively, a local improvement algorithm such as two-point swap, relocate, and 2opt makes some changes in that route resulting in an improved feasible solution with less cost (Groër, Golden, & Wasil, 2010).

Recent studies show that machine learning, especially Deep Reinforcement Learning (DRL), can automatically learn better heuristics than the one designed by a human for solving challenging NP-hard problems (Drori et al., 2020; Liu, Chang, & Tseng, 2020; Vesselinova, Steinert, Perez-Ramirez, & Boman, 2020). DRL has achieved a remarkable result in solving VRPs (Vesselinova et al., 2020). However, the developed DRL models cannot be trained for a large number of customers because the state space and action space expand exponentially as the number of customers increases, making it hard for the DRL to learn a useful routing policy. Current deep learning approaches for CVRP have strongly emphasized improving the learning phase to slightly outperform the previous learning model for non-practical VRPS. Most recent works ignored the scalability and generalizability aspects as they solve small VRPs with up to 100 customers using manually generated instances that ignore the complex nature of VRP (Vesselinova et al., 2020). On the contrary, few attempts utilize DRL for solving large TSP. For instance, Ma, et al., (2019) developed a graph pointer network model and trained it using an early stopping method to increase generalizability to solve large-scale TSP problems. Xu et al., (2021) developed an attention-based DRL model with enhanced node embedding to address different types of routing problems. Their work showed the generalization ability of attention-based models to address large TSP with 1000 customer nodes.

This paper introduces a two-stage DRL framework to address the previous challenges for large CVRP. A variant of DRL called the Attention model (AM) (Kool, Van Hoof, & Welling, 2018) is trained using the early stopping rule in the first stage. In the second stage, the trained DRL is combined with 2-opt local improvement heuristics to further improve the results. Investigating the generalizability of DRL and the effect of hybridization has not been considered in the previous research (François, Cappart, & Rousseau, 2019). Most of the recent works focus on small-sized CVRPs and other improvement techniques such as beam-search and sampling during the decoding to enhance the quality of the solutions (François et al., 2019). These methods work well for small-size problems only since they can provide a set of diverse solutions but fail when the size of the problem increases. Therefore, we choose a greedy decoding strategy during the testing before applying the 2-opt algorithm. The AM is very fast in the greedy decoding setting since it is specifically designed for it. As pointed out in (Kool et al., 2018), the AM can benefit from a post-processing procedure to handle the intercrossed route developed by the agent. Thus, we apply 2-opt in order to find an appropriate balance between solution quality and execution time when solving practical CVRP. Extensive experiments are carried out to evaluate the proposed methods. Different OR-tools metaheuristic strategies are used to conduct a competitive and fair experiment. As opposed to the previous learning model, the proposed method is tested on several classical CVRP benchmark instances developed by (Uchoa et al., 2017). These instances are generated to fill the gap between the real world and simulation and pose a significant challenge even for modern heuristics and metaheuristic (Uchoa et al., 2017). Results show that the hybrid DRL model outperforms the Google OR-tools with higher solution quality and shorter execution time.

This paper is organized as follows. Section 2 defines the CVRP and describes the selected DRL architecture. In Section 3, we explain the experiment in detail. Section 4 concludes this paper and highlights future work.

2. Deep Reinforcement Learning for CVRP

2.1 Problem Formulation

This paper focuses on CVRP which is a generalization of the TSP, where there is one depot and set of **n** customer points to be served. In a typical CVRP instance, the input $\mathbf{X = \{x_0, \ldots, x_n\}}$ is a set of nodes that represents the depot $\boldsymbol{X_0}$ and customers $\mathbf{X_i}$, $\mathbf{i \in \{1, \ldots, N\}}$. For each input point $\mathbf{X_i}$, it has two elements $\mathbf{(c_i, d_i)}$, where $\mathbf{(c_i = (x_i, y_i))}$ is a 2-dimensional coordinate of customer **i** in Euclidean space, and $\mathbf{d_i}$ is the corresponding customer demand ($d_0 = 0$). Several vehicles are needed to serve the customer demand. All vehicle starts and ends at the depot $\boldsymbol{X_0}$. Each vehicle has a limited capacity $\boldsymbol{Q}$ and can serve a set of customers in each route as long as the total customer demand does not exceed the capacity. In this research, all vehicles are considered to be homogenous, where all vehicles have the same capacity. The traveling cost $\boldsymbol{C_{ij}}$ is the cost and proportional to the distance traveled by the vehicle from customer **i** to customer **j**, with $\mathbf{i ; j \in \{1,\ldots, N\}}$. The solution of the CVRP is represented by a set of routes that satisfies the constraints. This solution can be viewed as a sequential decision problem, where the solution sequence is $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t)$ and $\mathbf{t \in \{1, \ldots, N\}}$ represents the vertices of every solution which starts and ends at the depot. Unlike TSP, the solution to CVRP may have varying sequence lengths. This is due to the fact that the vehicle may have to return to the depot several times to refill, even with the same number of customers.

Given a problem graph X, the DRL model uses the attention-based encoder-decoder architecture from Kool et al., (2018). The encoder extracts the structural features of graph X for all input nodes. Then, the decoder integrates the outputs of the encoder with a problem-specific masking scheme and context to produce the solution sequence**.** This process starts from a random node in the graph and predicts a probability distribution over nodes, one node for one vehicle at each timestep. A node

with the highest probability is chosen and attached to the end of the partial solution. When the partial solution has been constructed, it cannot be changed, and the problem at that time is to find a path from the last node for each vehicle's partial solution through all unvisited nodes to the depot. The entire solution of a CVRP is denoted by $\boldsymbol{\pi}$; our objective is to minimize the total routing length or cost L as follow:

$$L(\pi|X) = \sum_{t=1}^{n-1} \left\| c\,\pi_{(t)} - c\,\pi_{(t+1)} \right\|_2 , \qquad (1)$$

Where $\| . \|_2$ denotes ℓ_2 norm.

Therefore, the DRL model, parameterized by $\boldsymbol{\theta}$ and given a problem graph $\mathbf{X}$, needs to define a stochastics policy $\mathbf{p_\theta(\pi|X)}$ for selecting a solution sequence $\boldsymbol{\pi}$ that can minimize the total routing length $\mathbf{L}$. This stochastics policy can be factorized as follows:

$$p\theta(\pi_t|X) = \prod_{t=1}^{N} p\theta(\pi_t|X, \pi_1{:}\,\mathrm{t}-1) \qquad (2)$$

As a graph-based method, this stochastics policy can scale to larger instances without retraining for every new problem instance. Therefore, we can use the trained model to generate a fast and quality solution as a sequence of consecutive actions for new problems.

2.2 Encoder Framework

In the encoder framework, the graph attention network is used to provide a mapping from raw nodes features space to a richer embedding in the context of a graph. First, it computes the initial node embedding $\boldsymbol{h_i^{(0)}}$ for each input node $\boldsymbol{x_i}$ through a linear transformation with learnable parameters $\boldsymbol{W}$ and $\boldsymbol{b}$. Separate parameters $\boldsymbol{W_0}$ and $\boldsymbol{b_0}$ are used to compute the initial embedding $\boldsymbol{h_0^{(0)}}$:

$$\boldsymbol{h_i^{(0)}} = \begin{cases} W\,x_i + b & if \neq 0 \\ W_0 x_i + b_0 & if = 0 \end{cases} \qquad (3)$$

The initial node embeddings are updated using multiple attention layers $\boldsymbol{N}$. Each attention layer consists of two sub-layers: a multi-head attention sublayer and a node-wise fully connected feed-forward sublayer. The multi-head attention is used to combine and update the features of the nodes by extracting different types of information from all others. It can be interpreted as a weighted message passing between the nodes in a graph of layers. In the layer $\boldsymbol{\ell \in \{1,\ldots,N\}}$, $\boldsymbol{h_i^{(\ell)}}$ represents the node embedding of each node i, and the output $\boldsymbol{\{h_0^{(\ell-1)},\ldots, h_n^{(\ell-1)}\}}$ of the previous layer $\boldsymbol{\ell\text{-}1}$ is the input of the current layer $\boldsymbol{\ell}$. Then, the multi-head attention vector $\boldsymbol{MHA_i^{(\ell)}}$ combines a set of values $\boldsymbol{v}$ of each node $\boldsymbol{i}$ regrouped based on the similarity between their queries and linked keys, which can be computed as follow:

$$q_{im}^{(\ell)} = W_m^Q h_i^{(\ell-1)},\ k_{im}^{(\ell)} = W_m^k h_i^{(\ell-1)},\ v_{im}^{(\ell)} = W_m^V h_i^{(\ell-1)} \qquad (4)$$

$$u_{ijm}^{(\ell)} = \left(q_{im}^{(\ell)}\right)^T k_{jm}^{(\ell)} \qquad (5)$$

$$a_{ijm}^{(\ell)} = \frac{e^{u_{ijm}^{(\ell)}}}{\sum_{j=0}^{n} e^{u_{ijm}^{(\ell)}}} \qquad (6)$$

$$h_{im}^{(\ell)} = \sum_{j=0}^{n} a_{ijm}^{(\ell)}\, v_{jm}^{(\ell)} \qquad (7)$$

$$MHA_i^{(\ell)}\left(h_0^{(\ell-1)},\ldots,h_n^{(\ell-1)}\right) = \sum_{j=0}^{n} W_m^O\, h'^{(\ell)}_{im} \qquad (8)$$

Where, $\mathbf{m \in \{1,\ldots, M\}}$ represents the number of heads in each single attention layer, the $\boldsymbol{q_{im}^{(\ell)}, k_{im}^{(\ell)}, v_{im}^{(\ell)}}$ *are* the query, key and value vectors for each node by projecting the embedding $\boldsymbol{h_i^{(\ell-1)}}$. MAH takes the input as a set matrix of queries $\boldsymbol{q_{im} \in R^{dk}}$ and matrix keys of $\boldsymbol{k_{im} \in R^{dk}}$, and matrix values $\boldsymbol{v_{im} \in R^{dv}}$. It splits the computation onto M parallel to calculates the compatibility $u_{ijm}^{(\ell)}$ between the respective keys $\boldsymbol{K_{jm}^{(\ell)}}$ for node *j* and queries $\boldsymbol{q_{im}^{(\ell)}}$ for node $\boldsymbol{i}$.

In the MHA, d_h =128 is the vertical dimension of the $\boldsymbol{h_i^{(\ell)}}$, which is used to scale the dot-products for normalized similarity measures and prevent an overflow of numerical calculations. For each head $\boldsymbol{M}$, the compatibility score is calculated by using the method of scaled dot-product. Then, the attention weight $\boldsymbol{a_{ij} \in [0, 1]}$ is computed using softmax. This weight is

used to calculate the output of the attention $\boldsymbol{h_i'^{(\ell)}}$. And the final output $\boldsymbol{MHA_i^{(\ell)}}$ is computed with $\boldsymbol{W_m^O}$. MHA sublayer can be used as a graph encoder. However, it will not be able to perform any deep learning task and will lack the capability to represent complex nonlinear functions. Therefore, in the transformer architecture (Vaswani et al., 2017), the MAH sublayer is followed by a simple, fully connected feed-forward sublayer (FF) with

dimension d_{ff} to add nonlinear relation to the learning mechanism. The FF sublayer consists of 2 layers (hidden layer and output layer) with skip connection to compute node-wise projections. For each node $\boldsymbol{i}$ in the FF network:

$$\hat{h}_i^{(\ell)} = tanh\,(h_i^{(\ell-1)} + MHA_i^{(\ell)}(h_0^{(\ell-1)}, \dots, h_n^{(\ell-1)})), \tag{9}$$

$$FF(\hat{h}_i^{(\ell)}) = W_1^F + ReLu\ \ (W_0^F \hat{h}_i^{(\ell)} + b_0^F\,,) + b_1^F\,, \tag{10}$$

$$\hat{h}_i^{(\ell)} = tanh\,(h_i^{(\ell)} + FF(\hat{h}_i^{(\ell)})), \tag{11}$$

Finally, after $\boldsymbol{N}$ attention layer, the encoder computes an aggregated (graph embedding) $\boldsymbol{\bar{h}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} h_i^N}$ of all nodes i as the mean of the final node embedding $\boldsymbol{h_i^N}$. Both of $\boldsymbol{h_i^N}$ and $\boldsymbol{\bar{h}^{(N)}}$ are used as input to the decoder.

2.3 Decoder Framework

In the decoder, at each time step t, one customer node is selected to visit based on the embedding from the encoder and pervious partial solution $\boldsymbol{\pi_{(t-1)}}$. The objective is to find a solution that minimizes the total routing length **L** in Eq(1). To approach this problem with RL, the following subsequent subsections provide the basic elements of the RL.

2.3.1 Agent

The standard AM is a single-agent approach that is initially designed for TSP. CVRP is reformulated as a single vehicle returning to the depot multiple times. In this formulation, partial solutions are sequentially built one after the other, and there is no way to represent multiple partial solutions in parallel. Although this representation is simple, it works effectively for CVRP since all vehicles have the same capacity. At each time step, the agent is in a given state and chooses a sequence of actions action $\boldsymbol{\pi_t}$. The action chosen dynamically affects the environment and hence changes the state for the agent. Receiving reward $\boldsymbol{-L}$ from the environment, the goal is to maximize the cumulative rewards by learning a good policy.

2.3.2 State

Each state in the decoder includes dynamic and static features. Static feature, which remains constant all the time, includes the location and demand for each customer and the maximum capacity of the vehicle. These static features are problem-specific since they stay unchangeable across solutions. Dynamic features are based on the running history (current/last location), which includes the remaining capacity $\boldsymbol{Q'}$ and its current position at time step *t*. The static features represent the initial state of the environment, while the agent state is represented by the dynamic features. The environment state is embedded by the graph embedding $\boldsymbol{\bar{h}^{(N)}}$computed by MHA. During the decoding, the $\boldsymbol{\bar{h}^{(N)}}$ is augmented with a special context vector to represent the vehicle decoding context $\boldsymbol{h_c^{(N)}}$ at each time step as:

$$\boldsymbol{h_c^{(N)} = \begin{cases} [\bar{h}^{(N)}\,; h_{\pi_{t-1}}^{(N)}; \hat{Q}_t\,] & t > 1 \\ [\bar{h}^{(N)}\,; h_0^{(N)}; \hat{Q}_t\,] & t = 1 \end{cases}} \tag{12}$$

Where [;] is a concatenation operator, $\boldsymbol{h_{\pi_{t-1}}^{(N)}}$ is the embedding of the node chosen at time step t-1, $\boldsymbol{\hat{Q}_t}$ is the remeaning capacity for a vehicle after visiting customer $\boldsymbol{i}$ which is updated as follow:

$$\boldsymbol{\hat{Q}_t = max\,(\hat{Q}_t - d_i)} \tag{13}$$

2.3.3 Action

The action $\boldsymbol{\pi_t}$ specifies the next destination at current time t for the vehicle. Similar to the encoder, the context vector $\boldsymbol{h_c'^{(N)}}$ is computed using the MHA as:

$$\boldsymbol{h_c'^{(N)} = MHA\,(h_c^{(N)})} \tag{14}$$

At each construction step, the decoder keeps track of the running history in order to construct a feasible solution. Any node that violates the CVRP constraints is masked. Therefore, any customer's node that have been already visited or whose

demand is greater than the remaining capacity of the vehicle is masked by setting $\boldsymbol{u_{j,t}} = -\infty$. The depot can be visited many times but not in two consecutive time steps. Finally, the decoder computes the probability $p\theta(\pi_t|X) = \prod_{t=1}^{T} p\theta(\pi_t|X, \pi_1 : \mathrm{t} - 1)$ of selecting the next node at each time step for with single head layer (M=1, so $d_k = d_h$):

$$q_c = W_1^Q h_c\ ,\ k_j = W_1^Q h_j^{(N)}, \qquad (15)$$

$$u_j = C.tanh\left(\frac{q_c^T k_j}{\sqrt{d_k}}\right), \qquad (16)$$

$$p\theta(\pi_t = x_j|X, \pi_{1:\mathrm{t}-1}) = \frac{e^{u_j}}{\sum_{j\prime=0}^{n} e^{u_{j\prime}}} \qquad (17)$$

where C is used for clipping the results within [-C, C] (C=10) using tanh.

2.3.4 Reward

When all demands are satisfied, the negative of the objective function L is received as a reward. As the cumulative reward increases, the maximum length of all routes is minimized by defining the reward as the negative total route length in the Euclidian space:

$$-L(\pi|X) = \sum_{t=1}^{n-1} \left\| C\,\pi_{(t)} - C\pi_{(t+1)} \right\|_2 \qquad (18)$$

2.4 Model Training

DRL only needs the reward signal to train the network. The DRL model, parameterized by $\boldsymbol{\theta}$ and given a problem graph $\mathbf{X}$, needs to be trained to find a stochastics policy $\mathbf{p\theta(\pi_t|X)}$ for selecting a solution sequence $\boldsymbol{\pi}$ that can minimize the expected total routing length of $\mathbf{L(\pi|X)}$:

$$J(\theta|X) = E_{\pi \sim P_\theta(.|X)} L(\pi|X). \qquad (19)$$

The policy gradient REINFORCE (Williams, 1992) is used to train the parameters:

$$\nabla_\theta J(\theta|X) = E_{\pi \sim P_\theta(.|X)}[(L(\pi|X) - b(X))\nabla_\theta\, log\, P_\theta(\pi|X)], \qquad (20)$$

Where b(X) is the baseline for estimating the route length $E_{\pi \sim P_\theta(.|X)} L(\pi|X)$. A good baseline can reduce the gradient variance and accelerate the learning process. The greedy rollout baseline is used to construct a greedy solution for the problem in which the agent selects the best action with maximum probability at each time step. This baseline b(X) is updated at the end of each epoch only if the difference in routing lengths for candidate parameters and a baseline is statistically significant based on the t-test with $\alpha = 0.05$. During the training process, batch $\mathbf{B}$ instances $x_1, x_2, \ldots x_B$ is drawn from the same distribution S to train the agent. Monte Carlo sampling is used to approximate the gradient in Eq(20):

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} [(L(\pi_i^s|X_i) - L(\pi_i^*|X_i))\nabla_\theta\, log\, P_\theta(\pi_i^s|X_i)], \qquad (21)$$

Where π_i^s and π_i^*, are the sample rollout and greedy rollout solutions of instance X_i. If $L(\pi|X) - b(X)$ is negative, then the sampled solution is better than the greedy baseline, causing the solution to be updated, and vice versa. Hence, the agent can effectively solve the routing problem by learning a good policy.

3. Experiments and Analysis

3.1 Setup and Hyper-Parameters

Extensive experiments are conducted to investigate the performance of the proposed model in addressing the large and practical CVRP. This research generated the training instances following (Nazari, Oroojlooy, Snyder, & Takáč, 2018), where the node locations (x_i, y_i) and customer demands di are randomly generated from a uniform distribution. The coordinates

of the depot and 100 customers are randomly generated in the unit square [0, 1] × [0, 1]. It is assumed that the traveling cost between two nodes *Cij* is proportional to the Euclidean distance. The demand of each customer is discrete and randomly generated from {1, … ,9}; the capacity of vehicle Q is set to 50 units. The batch size B = 128 and learning rate η = 0.0001. In the encoder, we set the initial node embedding size d_h=128, the number of attention heads M=8, and the number of attention layers N= 3. Finally, the DRL model was implemented in Python and realized using Pytorch. All experiments were conducted on a workstation configured with AMD Ryzen 5 processor and GeForce RTX2060 GPU

3.2 Early Stopping and Model Selection

Before evaluating the model on a real-world dataset, it is necessary to ensure its generalizability. In order to increase the generalization ability of the AM to larger-scale CVRP, CVRP100 is trained using an early stopping technique to avoid overfitting on small problems with 100 customers. For this process, we use 1000 synthetic instances generated on the fly to initially evaluate the performance of the learned policy on different levels of early stopping. During the training of CVRP100, the model is stopped at a specific number of epochs and used the trained model to solve larger CVRP of n= {200, 500, and1000}. In each epoch, 10000 batches of size B are processed (Total instances = 128*10000 per epoch) to keep training time traceable. Since Sampling and beam search struggles with larger instances, we use greedy decoding when testing the trained CVRP100. The comparison results between CVRP100 performance on various early stopping epochs is shown in table 1. The results are the average total route length for all 1000 instances. The gap to the OR-Tools solution is also reported in Table 1. We use the Automatic setting (Auto) for OR-tools to allow the solver to choose the best solution strategy. For the stopping rule, we follow (Zhao, Mao, Zhao, & Zou, 2020) and set the search time limit for each problem instance to 30*n *ms*, where n represents the number of customers.

Table 1. Average total routing length obtained by CVRP100 on different epochs.

Epoch		10	30	50	100	OR-Tools
N=200	Tour Cost	29.28	27.54	27.58	28.98	26.43
	Gap	9.73%	**4.03%**	4.17%	8.80%	0.00%
N=500	Tour Cost	73.02	68.72	68.75	69.81	65.73
	Gap	9.98%	**4.35%**	4.39%	5.84%	0.00%
N=1000	Tour Cost	145.25	141.38	141.75	142.01	129.51
	Gap	10.84%	**8.40%**	8.63%	8.80%	0.00%
N=2000	Tour Cost	313.25	297.71	297.94	299.03	253.84
	Gap	18.97%	**14.74%**	14.80%	15.11%	0.00%

The CVRP100 model trained for 30 epochs is selected to solve the large-scale classical benchmarks problem. Using Earley stopping rule, the AM-CVRP100 model appears to scale well when tested on instances generated from a uniform distribution. However, it demonstrates difficulty scaling up to instances with N= 2000 customers. Therefore, we limit the testing of the trained CVRP100 to instances with up to 1000 customers.

3.3 Hybridization strategy

In some cases, the learned policy by the agent creates a suboptimal routing plan where different routes are intercrossed. Therefore, the heuristics learned by the DRL is combined with a 2-opt algorithm to quickly improve the outputted solutions without increasing the execution time during the inference. The 2-opt was proposed by Croes, (1958) to solve TSP in 1958. The main idea of the 2-opt is to identify two intercrossed routes and then rearrange them to form a new solution with less distance. This process terminates when there is no further improvement in tour length. Every route generated in CVRP can be considered a single TSP that can benefit from running 2-opt when the order of vehicle stops is suboptimal. However, the quality of the initial solution obtained by the trained CVRP model affects the quality of the improved solution. For example, if the agent performs more trips than necessary to the depot, running 2-opt will not yield a significant improvement in this case. For this reason, evaluating the results before and after running 2-opt is critical for addressing the learning ability of DRL and the improvement rate obtained by 2-opt.

3.4 Real-world Benchmarks and Baselines

To further investigate the robustness of the trained CVRP100 model, we test the CVRP100 trained for 30 epochs on several classical benchmark instances created by (Uchoa et al., 2017). These instances cover different characteristics (e.g., capacity of vehicles, customer and depot locations, route length, distribution of demands) and represents a significant challenge for modern heuristics and metaheuristics. For a detailed description of the instance characteristics, we refer the reader to (Uchoa et al., 2017). To evaluate the performance of the proposed hybrid DRL model on the selected instances, the solutions are compared against the well-known metaheuristic's solver OR-Tools. OR-Tools has different strategies for finding the initial solution, which can be incorporated into different metaheuristics algorithms. The recommended setting uses the Path Cheapest Arc strategy heuristics with *Guided Local Search* (GLS) metaheuristic. We use different settings for OR tools: 1) we use the automatic setting and allow the solver to automatically choose the best combination of heuristics and metaheuristics based on the targeted problem. 2) we fix the initial solution strategy and use Simulated Annealing (SA) and Tabu Search (TS). OR-Tools library for VRP includes 12 different heuristics, each of which can be incorporated into more search exhausting metaheuristics algorithms (Kruk, 2018) (Surana, 2019). Hence, testing all combinations is out of the scope of this paper. The run time for each instance is set to 30*n *ms*. It should be noted that most of the previous DRL approaches in the literature did not specify or mention the stopping criteria or the setting used for OR tools.

3.5 Results

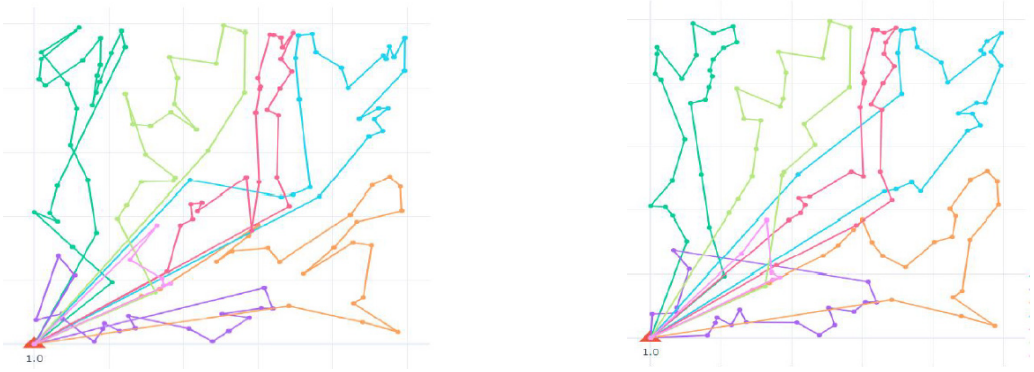
The DRL testing results are all obtained by using the final learned policy using greedy decoding. We run the policy trained on CVRP100 on 13 instances with sizes between 143 to 979. Each instance has a different vehicle capacity and is generated using various customer locations, depot locations, and demand distribution. The best-known solution for these instances was obtained using sophisticated exact and non-exact methods. In most large CVRP instances, long-running time was required to obtain such a quality solution. Therefore, we do not compare our approach to these highly specialized algorithms. Instead, the obtained best-known solutions are used to evaluate the performance of the adopted approach. Table 2 shows the results of the hybrid DRL model compared to different OR-tools baselines.

Table 2. Results on CVRP benchmarks

Instance	DRL (greedy)	DRL (greedy) & 2opt	OR-tools (Auto & Auto)	OR-tools (Path Cheapest Arc & GLS)	OR-tools (Path Cheapest Arc & SA)	OR-tools (Path Cheapest Arc & TS)	Best Known Results
X-n143-k7	20210 (0.2s)	17431 (0.3s)	17392	17451	**17332**	16909	15700
X-n261-k13	33110 (0.3s)	30231 (0.7s)	**29373**	29671	30143	29402	26558
X-298-k31	40181 (0.4s)	**37900** (1s)	40478	41476	41481	41505	34321
X-n303-k21	25935 (0.4s)	**23838** (1.2s)	24193	24457	24779	24820	21744
X-n449-k29	64433 (0.7s)	**60436** (2.4s)	60821	60853	60870	60880	55358
X-459-k26	30480 (0.7s)	27800 (2.5s)	**27067**	27776	26995	27088	24181
X-n513-k21	33009 (1s)	28026 (3.1s)	27999	28184	28128	**27591**	24201
X-n613-k62	72433 (1.4s)	69989 (3.6s)	**69966**	73460	69999	70011	59778
X-685-k75	80743 (1.8s)	**78770** (3.8s)	85256	92686	91786	83905	68425
X-n716-k35	55797 (1.9s)	**50110** (4.4s)	58782	58930	50393	51125	43525
X-766-k71	143127 (2.3s)	133019 (5.3s)	135070	**131825**	133199	135447	114683
X-n895-k37	69110 (3s)	60274 (6.8s)	60442	**59993**	61746	61429	54172
X-n979-k58	136010 (3.5s)	**127260** (8.6s)	130673	127297	127545	128519	119194
Average Gap	23.26%	12.79%	14.21%	14.30%	14.77%	14.71%	0.00%

The results show that the hybrid DRL (AM-CVRP100 + 2opt) achieves the lowest average gap of 12.79% with significantly less execution time. The smallest instance was solved in 0.2 seconds, while it took 8.6 seconds to solve the largest instance. Also, our method outperformed OR tools in 6 out of 13 instances tested. These results are promising since OR-tools is a highly specialized solver with fine tunned parameters. Our best results for each instance are produced by using the hybrid DRL only. On the other hand, the best results obtained by OR-tools are generated using different metaheuristics strategies.

Although using the automatic strategy shows better performance than other strategies, it is not clear which strategy works the best for all the instances. The learned policies by the DRL trained with 100 customers are not too far from OR-tools. We noticed that The DRL generalizes well when tested on instance from a uniform distribution, but it has difficulty generalizing when tested on the problem out of the distribution. By visualizing the solution, we found that the main reason for the weak generalization is not the number of generated routes but the order of customer visits in each individual route. It appears that the agent learned how to generate an appropriate number of routes in sequence based on the vehicle capacity but struggles with the customer visits in these routes. Therefore, each route is treated as a single TSP in the hybrid DRL and improved with 2-opt. Around 45% decrease in the average gap is observed when running the 2-opt on the top of the DRL solution. The 2-opt algorithm terminates when TSP tours cannot be improved by 2-change anymore. Figure 1 shows a visualization example of the solution obtained by DRL vs. Hybrid DRL for X-n143-k7



(a) DRL solution (b) Hybrid DRL solution

Figure 1. Visualization for X-n143-k7 instance

4. Conclusion

This paper proposes a two-stage DRL framework to approximately solve the large CVRP. First, a DRL using an attention mechanism is trained with early stopping to avoid overfitting on the small instances. Then, the trained model is combined with a local search 2-opt to handle the drawback of the learned policy. The proposed approach is tested on classical benchmark instances to evaluate its applicability for solving practical CVRP. The experimental results on several classical CVRPs showed that the hybrid DRL model not only improves the solution quality but also outperforms the optimization solver OR-tools with less execution time.

Concerning future studies, it is interesting to see how the proposed framework can improve the solution when used for generating initial solutions for OR-tools or other metaheuristics. Also, future problems with more realistic constraints, such as CVRP with time-window and multiple depots, will be considered.

5. References

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research, 6*(6), 791-812.

Drori, I., Kharkar, A., Sickinger, W. R., Kates, B., Ma, Q., Ge, S., . . . Udell, M. (2020). Learning to solve combinatorial optimization problems on real-world graphs in linear time. *arXiv preprint arXiv:.03750*.

François, A., Cappart, Q., & Rousseau, L.-M. J. a. p. a. (2019). How to evaluate machine learning approaches for combinatorial optimization: Application to the travelling salesman problem.

Groër, C., Golden, B., & Wasil, E. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation, 2*(2), 79-101.

Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems! *arXiv preprint arXiv:08475*.

Kruk, S. (2018). *Practical Python AI Projects: Mathematical Models of Optimization Problems with Google OR-Tools*: Apress.

Labadie, N., Prins, C., & Prodhon, C. (2016). *Metaheuristics for vehicle routing problems*: John Wiley & Sons.

Liu, C.-L., Chang, C.-C., & Tseng, C.-J. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access, 8*, 71752-71762.

Ma, Q., Ge, S., He, D., Thaker, D., & Drori, I. (2019). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv :04936*.

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:04240*.

Surana, P. (2019). Benchmarking Optimization Algorithms for Capacitated Vehicle Routing Problems.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research, 257*(3), 845-858.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:.03762*.

Vesselinova, N., Steinert, R., Perez-Ramirez, D. F., & Boman, M. (2020). Learning Combinatorial Optimization on Graphs: A Survey With Applications to Networking. *IEEE Access, 8*, 120388-120416.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research, 231*(1), 1-21.

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

Xu, Y., Fang, M., Chen, L., Xu, G., Du, Y., & Zhang, C. (2021). Reinforcement Learning With Multiple Relational Attention for Solving Vehicle Routing Problems. *IEEE Transactions on Cybernetics*.

Zhao, J., Mao, M., Zhao, X., & Zou, J. (2020). A hybrid of deep reinforcement learning and local search for the vehicle routing problems. *IEEE Transactions on Intelligent Transportation Systems*.