LEARN TO DESIGN HEURISTICS USING REINFORCEMENT LEARNING TO SOLVE VRP

Jaswanth Badvelu

IENG 6923 Distribution Management

12/7/20

Table of Contents

LEARN TO DESIGN HEURISTICS USING REINFORCEMENT LEARNING TO SOLVE VRP

1 ABSTRACT

In recent years machine learning is evolving at a phenomenal rate and can tackle tough problems on its own. The recent research work in the field of combinatorial optimization shows that machine learning has the potential to learn and design heuristics better than the traditional heuristics designed by humans. In this project, a reinforcement model with dynamic encoder-decoder architecture is developed that learns to design its heuristics based on the data to solve a large-scale vehicle routing problem with optimality. The trained models produce the near-optimal solution instantly, without the need to retrain the models. When compared, with other heuristic approaches like the Savings Clarke wright algorithm which is implemented in Google's Operation research tools this reinforcement model outperformed them. This proposed model can be easily extended to solve other variants of VRP problems like multi depot and VRP with Time windows.

2 INTRODUCTION

Millions of goods are being transported every day, in the year 2018 alone the United States business logistics reached 1.6 trillion dollars (8% of GDP that year), which is why reducing the cost of transportation of goods and services is very crucial for every company. Most of the companies use road transportation with limited vehicle capacity to deliver goods to multiple locations. This transportation route can be optimized using the Capacitated vehicle routing problem, which is one of the variants of the vehicle routing problem. Determining the optimal solution of VRP is an NP-hard type problem which is one of the hardest optimization problems to solve. Mathematical approaches like linear programming, branch, and bound algorithms can be used to find the optimal solution for small problems. Due to complexity, traditional optimization approaches cannot be used to solve large scale problems. The heuristics approach can be used to solve large-sized problems within a reasonable amount of but since heuristics are based on approximations only near-optimal solutions can be found.

State-of-the-art algorithms rely on handcrafted heuristics to find near-optimal solutions for solving large-sized problems. For some companies there is need to find optimal routes multiple times a day with different customer sizes. Handcrafted traditional heuristics approaches are too expensive and time-consuming to design, this issue can be resolved if we can design heuristics without human intervention. Neural networks and reinforcement learning can offer endless possibilities to the idea of designing heuristic algorithms without human intervention.

The proposed model uses dynamic attention model architecture with encoder-decoder architecture. The results indicate that the Dynamic attention model performs significantly better than the attention model. A uniformly generated random dataset with 10,000 samples is used to train the model network using the policy gradient method.

The model can be used to solve all the problems with various instances by training the model once.

Other than giving better optimal solutions some other advantages by using reinforcement models are once trained this model can be used to solve very large problems even up to 400 nodes and optimal solution can be found almost instantly without retraining the model every time. The same instance can be used to solve problems of different sizes. Whereas with other problems designing the heuristics can be challenging and the model can take several hours to find the optimal solution.

To test the validity of the developed model, the UPS store with a single depot is considered with multiple customers in the Vancouver region. Since the number of customers varies daily the model is trained to find the optimal route for 20,30,40,50 customer locations.

This model when compared with models that are already trained to solve a particular set of customers outperformed the saving algorithm heuristic approach. When applied on very large problems(100 customers) for which the model is not trained, the reinforcement model performed better and sometimes almost equal to the saving algorithm.

Using this Reinforcement model that learns to design own heuristics, instead of other traditional heuristic approaches can provide a significant advantage to the companies because there is no need to redesign the model every time the data input values are altered.

This reinforcement model is very appealing because of generalization, once the model for a medium instance is trained which will take less than 10 hours, the model can be used to find the optimal route instantly for even large instance problems.

In the case of small companies with no computing resources, the model can be trained up to 50 customers using free GPU resources like Google Colab, because of good generalization, this model can be used to find the optimal route for customers till 150 customers. The same model can be applied to solve even very large problems like 400 customers, but the performance of the model may be deterred for which solutions might not be near-optimal. Because the solution is generated instantly it is still good to use this model to solve large instance problems also if there is no other alternative that can provide an optimal solution in a reasonable amount of time.

3 RELATED WORK

Many researchers have previously worked on the application of neural networks to combinatorial optimization. One of the earliest proposals was made by [2] Hopfield (1990) by using Hopfield networks to solve small TSP, where he modified the network's energy function to match it with the TSP objective and used multipliers to penalize the violations of the problem's constraints. A limitation of this approach is that the model was sensitive to hyperparameter initialization.

The recent advancements in sequence-to-sequence learning enhanced the research work in the field of neural combinational optimization. [3] Bello (2016) showed the effectiveness of this approach by developing a neural combinatorial optimization architecture with an attention mechanism to solve TSP and the knapsack problem that uses reinforcement learning to optimize a policy modeled by a pointer network. Even though this model is very effective on combinatorial optimization problems like TSP and Knapsack this model cannot be applied to solve more complicated problems like VRP in which system representation changes over time. This model assumes that the system is static over time which is true in the TSP whereas in the case of VRP the demand should be updated every time a node is visited.

The limitations of using a pointer network for such a case are very difficult because at every node the model should remember how much demand is used and when the vehicle capacity becomes zero the model should recalculate the total structure. This issue becomes even more difficult during the backpropagation to use the gradient methods where the model needs to remember when the dynamic elements changed.

This issue is addressed by [4] Goa (2019) where he introduced an Actor-Critic algorithm to train the pointer networks without the need for supervised solutions. In this approach, every node is considered separate and cost (tour length) is used for an unbiased Monte-Carlo estimate of policy gradient. The proposed neural network model consists of an encoder in form of a modified version of Graph Attention Network where node embeddings and edge embeddings are integrated, and a GRU-based decoder rendering a pair of destroying and repair operators. [5] Kool (2020) proposed a model based on attention layers that shows that using the attention layer has significant benefits compared to the pointer network model.

A multi-head attention model was proposed by [6] Nazari (2018) with a simplified version of the pointer network with a set of embeddings that maps the inputs into a D- dimensional vector space instead of encoder in the RNN to solve split delivery VRP problems. However, in this model, the state of instance represented by the nodes is fixed as an 8-dimensional vector, but the state of instance should be updated dynamically according to the decision of the model made at different construction steps.

[7] Chen (2018) proposed a neural network design using Hierarchical Recurrent Graph Convolutional Network (HRGCN) as a large neighborhood search heuristic that learns the heuristics to solve very large problems. This approach uses Dynamic Partial Removal as a destroy heuristic, which provides an advantage to search across a large scale.

In the model proposed by [8] Deudon (2016). an explicitly forgetting mechanism model is introduced which only requires the last three selected nodes per step to construct a solution.

Despite many different approaches, all these model shares common methodologies where an encoder-decoder architecture with attention is designed to generate sequential decisions that construct the routes. The encoder is used to input the information to the nodes regarding demand and locations and the decoder to store the information and give output. And the policy gradient method is used to train the network. Figure 1 represents the encoder-decoder structure for VRP.

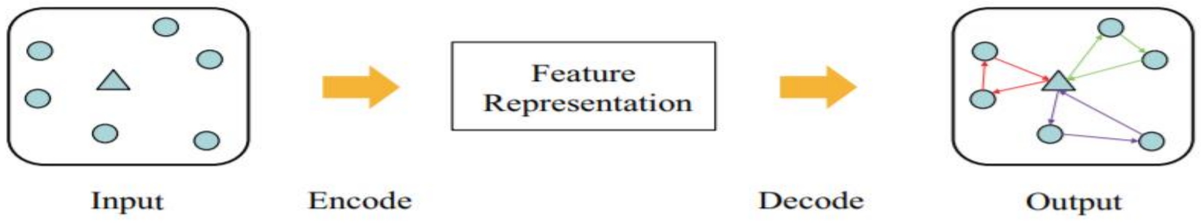


Figure 1. encoder-decoder structure for VRP models

4 BACKGROUND

Before proceeding to the model some information about concepts used in the model is provided.

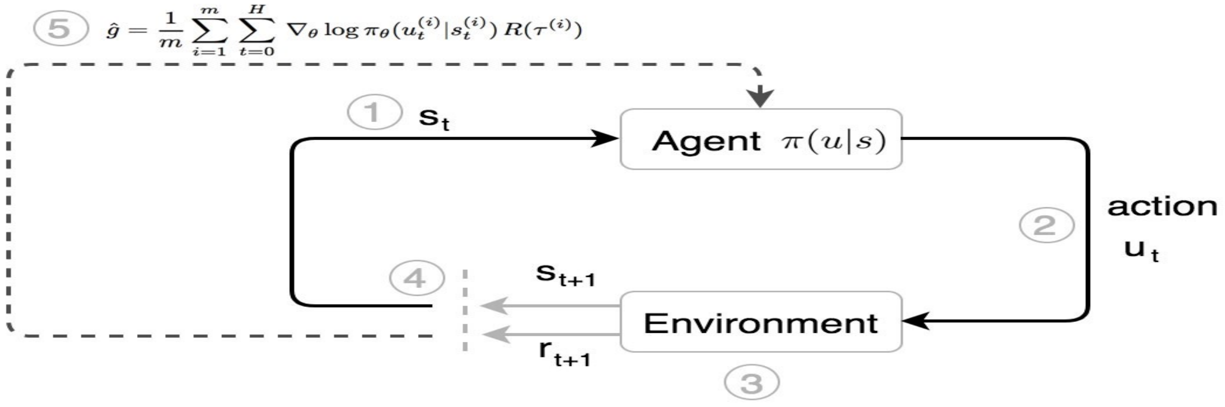
Reinforcement Learning

Based on Reinforcement learning systems, researchers have achieved some amazing feats recently like defeating world champions of Go and mastering multiple Atari games. Reinforcement learning consists of an agent that treats the problem of finding optimal or sufficiently good actions in the environment for a situation to maximize a reward. In other words, it learns from interactions. After every step, the agent is transitioned into the new state.

A human take actions based on observations. Similarly, the reinforcement system learns new things by observing the environment and uses the acquired knowledge to tackle the situation.

Policy Gradient

Policy gradient methods are frequently used algorithms in reinforcement learning. In policy gradient methods we train an agent to act based on observations. Figure 2. represents the agent-environment feedback loop [8]



5 $\hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\,R(\tau^{(i)})$
1 s_t
Agent $\pi(u|s)$
2
action u_t
4
s_{t+1}
r_{t+1}
Environment
3

Figure 2. The Canonical Agent Environment loop

1) Observe the state of the environment (s).
2) Take actions (u) based on the instinct (a policy π) on the state (s).
3) A new state is formed based on the action taken.
4) Further actions are taken based on the observed state.
5) After a trajectory of motions, the instincts are adjusted based on total rewards $R(\tau)$ received.

The policy gradient can be mathematically formulated as:

$$J(\theta) = E[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_\tau P(\tau; \theta)R(\tau)$$

And our objective is to find a policy θ that creates a trajectory τ that maximizes the expected rewards. $\max J(\theta) = \max \sum_\tau P(\tau; \theta)R(\tau)$

Graph Embeddings

Graphs consist of edges and nodes. Graph Embeddings contain node properties in a vector with a smaller dimension. Machine learning on the graphs is limited because the graph network can only use a specific set of statistics. Whereas in vector spaces many different approaches can be used. This provides an edge as vector operations are simpler and faster.

Savings Algorithm

The Clarke and Wright savings algorithm is one of the most known heuristics for VRP. It was developed by Clarke and Wright in 1964.

How does the savings algorithm work?

1. Assign individual vehicles for every job.
2. Calculate the maximum savings for putting two jobs together on the same vehicle if the demand constraints are met (saving one trip from the depot), if there are no savings, stop, otherwise proceed to step 3.
3. Implement the change and repeat step 2.

Step 2 can be done in two different methods:

1) **Best feasible merge (Parallel Version):** Determine where there exist two routes that have the feasibility to be merged.[9]
2) **Route Extension (Sequential Version):** Determine the savings that can feasibly be used to merge the existing route with another route.[9]

In this project, the saving algorithm is implemented using Google OR Tools and the performance is compared against the Reinforcement model to measure the optimality gap.

5 ATTENTION MODEL FOR CVRP

This project focuses on developing a reinforcement model for the Capacitated Vehicle Routing Problem. For the sake of simplicity, a problem with a single depot and multiple customers is considered.

5.1 Model Formulation

The capacitated Vehicle Routing problem is an extension of VRP in which vehicles with limited capacity need to visit various locations to pick up or deliver items. The problem is to minimize the cost for delivery or pick up of items in multiple locations, while never exceeding the capacity of the vehicles.

Let G = (V,E) be a graph with V = {1,2, …. ,n}being a set of vertices representing n customer locations with the depot located at vertex 1 and E being a set of undirected edges. With every edge (i,j) ∈ E, $i \neq j$ a non-negative cost c_{ij} is associated. This cost may, for instance, represent the distance between two customers i and j. Furthermore, assume there are m vehicles stationed at the depot that have the same capacity Q. Also, every customer has a certain demand q.

The CVRP consists of finding a set of vehicles routes such that

- Each customer in V is visited exactly once by exactly one vehicle
- All routes start and end at the depot
- The sum of customer demand within a route does not exceed the vehicle capacity
- The sum of costs of all routes is minimal given the constraints above

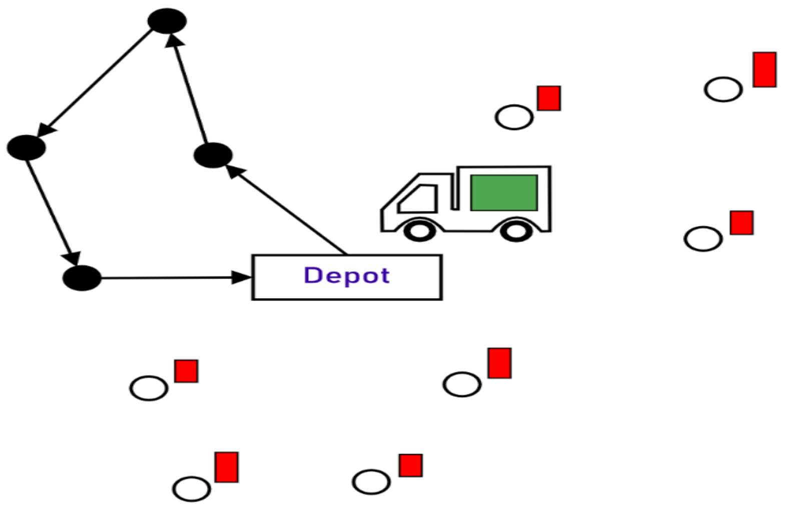


Figure 3. Capacitated VRP Model

Using encoder-decoder architecture will be very effective to solve VRP because this problem can be viewed as a sequential decision-making problem. An encoder is used to extract the structural features of the input for all the nodes. The solution is constructed incrementally by the decoder. The encoder layer acts to compress the latent information, whereas the decoder layer acts to reconstruct this information as accurately as possible. This forces the autoencoder to learn features to encode useful information into the hidden layer nodes. Autoencoders may be used to automatically extract high-level features which may also be called embeddings.

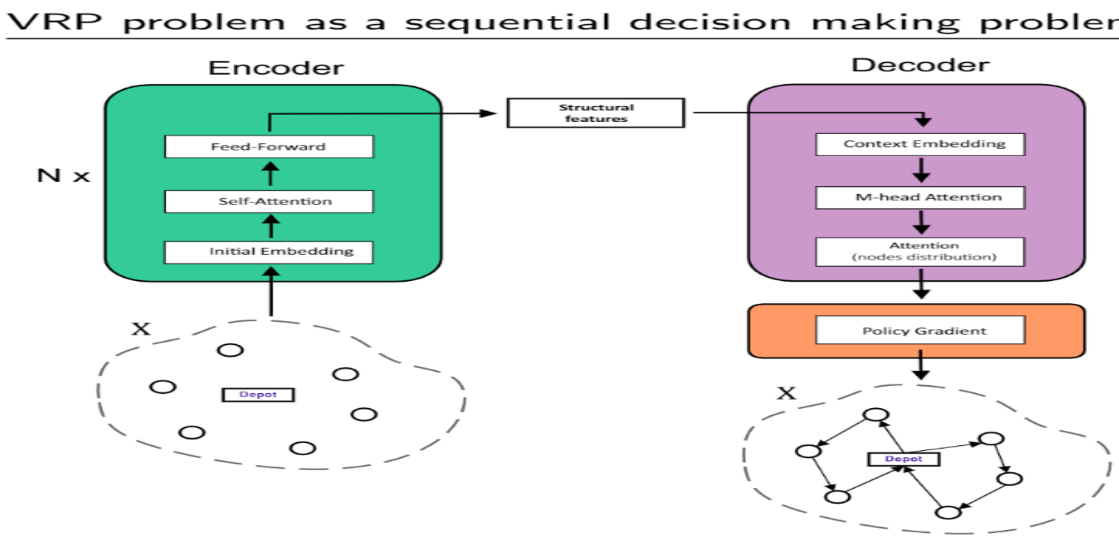


Figure 4. Encoder-Decoder Architecture

At each construction step, the decoder predicts a distribution over nodes, then one node is selected and appended to the end of the partial solution. The probability of the solution for graph instance X can be decomposed by chain rule as

$$p_\theta(\pi | X) = \prod_{t=1}^{T} p_\theta (\pi_t | X, \pi_{1:t-1})$$

5.2 Encoder

In the encoder, graph attention networks are used to encode nodes to an embedding in the context of the graph. First, all the input nodes are assigned with the location coordinates and demand (dx = 3), the dh-dimensional (d_h = 128) initial node embedding $h_i^{(0)}$ is computed through a linear transformation with learnable parameters W∈ $R^{d_h * d_x}$ and b ∈ R^{d_h} , separate parameters W_0 and b_0 should be used for the depot.

$$h_i^{(0)} = \begin{cases} Wx_i + b \ \text{ if } i \neq 0 \\ W_0 x_i + b_0 \text{ if } i = 0 \end{cases}$$

All the initial node embeddings are fed into the first layer of the graph attention network and updated with N = 3 attention layers. Embedding (hidden state) for each node V_i is obtained by a weighted sum of features of all nodes $V_{j\in \mathrm{N}(i)}$ in some neighborhood N(i). Weights a_i are calculated by the attention mechanism, representing the importance of each neighbor for a specific node.

Every layer contains two sub-layers: multi-head attention (MHA) sublayer and a fully connected feed-forward (FF) sublayer.

5.2.1 Multi-Head Attention:

Multi-Head Attention is used to extract different types of information. In the layer ℓ ∈ {1, . . . , N}, $h_i^{(l)}$ is denoted as the node embedding of each node i, and the output $\{h_0^{(l-1)}, \dots, h_n^{(l-1)}\}$ is of the layer ℓ - 1 is the input of the layer ℓ.

The linear project initial embeddings of the multi-head attention vector H ∈ $R^{batch*n*d_h}$ (query, key, value) can be computed by splitting the number of the head into M=8 to compute compatibility matrix A ∈ $R^{batch*M*n*n}$ for graph nodes Q = $W^Q h_i^{(l-1)}$, K = $W^K h_i^{(l-1)}$, V = $W^V h_i^{(l-1)}$.

The attention messages for each head can be computed with $W_m^O \in R^{d_h * d_v}$ ($d_k = d_v = d_h$/M = 16). and concatenated which results in the multi-head attention vector as

$$\mathrm{MHA}_i^{(l)} \ (h_0^{(l-1)}, \dots, h_n^{(l-1)}) = \sum_{m=1}^{M} W_m^O h_{im}'^{(l)}$$

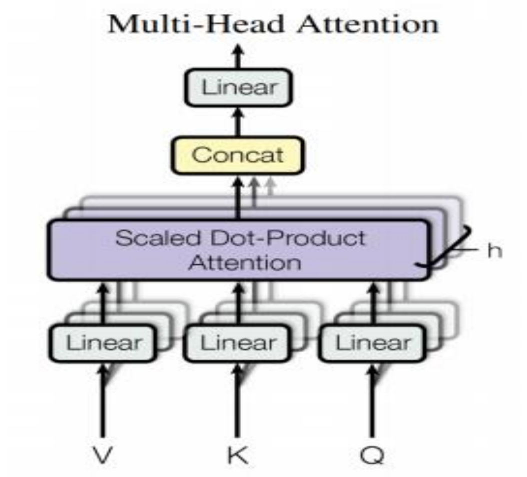


Figure 5. Multi-Head Attention sublayer

5.2.2 Feed Forward Sublayer

In this sublayer, for each node i, $h_i^{(l)}$ is computed by applying a fully connected feed-forward (FF) network with skip connections, $\ell \in \{1, \ldots, N\}$

$$h_i^{(l)} = \tanh\left(h_i^{(l-1)} + \mathrm{MHA}_i^{(l)}\ (h_0^{(l-1)}, \ldots, h_n^{(l-1)}) \right),$$

$$\mathrm{FF}(h_i^{(l)}) = \mathrm{W}_1^{\mathrm{F}}\mathrm{ReLu}(\mathrm{W}_1^{\mathrm{F}} h_i^{(l)}) + b_0^F) + b_1^F$$

$$h_i^{(l)} = \tanh\left(h_i^{(l)} + \mathrm{FF}(h_i^{(l)})\right)$$

After N layers we get the final node embeddings

$$h_i^N = ENCODE_i^N\ (h_0^N, \ldots, h_n^N)$$

5.3 Decoder

In decoder, at each construction step $t \in \{1,2, \ldots, T\}$, one node is selected to visit based on the partial solution $\pi_{1:\, t-1}$ and the embedding of each node. The policy is governed by two sequential attention layers in the decoder.

Multi-head attention layer: Query vector from context vector, Keys and Values from embeddings of nodes

Single head attention layer (Only compatibility) for probabilities: Add mask to attention: mask nodes that have been visited or have too much demand

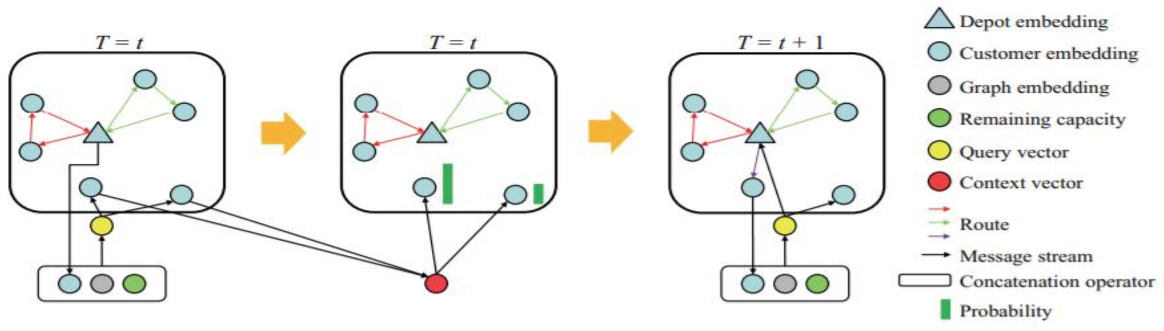


Figure 6. Decoder details at construction steps

The details of the decoder at the construction step t. At each construction step, concatenate mean graph embedding over all nodes, embedding of the previously selected node, and remaining capacity of the vehicle, the decoder predicts a distribution over nodes and selects one to visit.

$$h_c = \begin{cases} [h_c; h_0^N\ ; D_t] \text{ for } t = 1 \\ [h_t\ ; h_{\pi_{t-1}}^N\ ; D_t] \text{ for } t > 1 \end{cases}$$

Where [;] is a concatenation operator, $h_{\pi_{t-1}}^N$ is the embedding of the node selected at construction step t-1, D_t is the remaining capacity of the vehicle.

6 DYNAMIC ATTENTION MODEL FOR VRP

The solution is incrementally created by the decoder, at different construction steps, the state of the instance is changed, and the feature embedding of each node should be updated. Whenever the vehicle completes one journey, a new instance is created with the remaining nodes. Since some nodes are not visited the structure information should be changed and node features should be updated accordingly. In the dynamic encoder-decoder architecture the embedding of each node will be immediately recomputed when the vehicle returns to the depot.

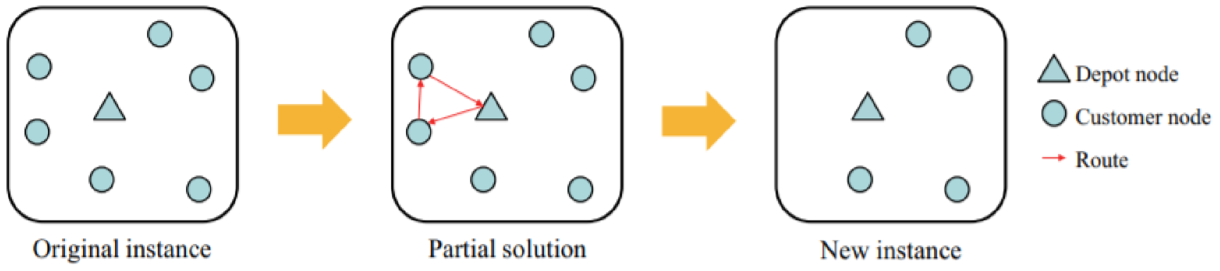


Figure 7. Dynamic Attention Model Working Process

Every time the vehicle is returned to the depot, the remaining nodes are considered as a new (smaller) instance (graph) to be solved.

The embeddings of the remaining nodes can be updated using encoder when the agent returns to the depot at construction step t as:

$$h_i^t = \begin{cases} ENCODE_i^N \ (h_0^0, \ldots, h_n^0) \text{ if } \pi_{t-1} = x_0 \\ h_i^{t-1} \text{ otherwise,} \end{cases}$$

Where, h_i^t is the embedding of node *i*, at construction step *t*, and the layer of *N* is omitted.

6.1 Dynamic Encoder-Decoder

In the vanilla encoder-decoder architecture of AM for VRP, the encoder is used only once, the embedding of each node is fixed, which only can represent the initial state of the input instance. Whereas in the Dynamic encoder-decoder architecture of AM-D, the encoder and decoder are used alternately to recode the embedding of each node and construct a partial solution.

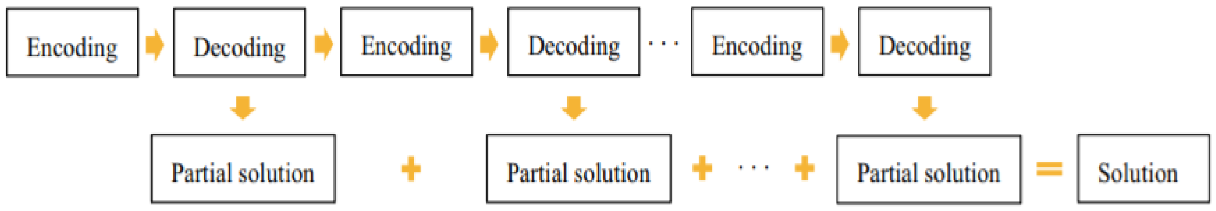


Figure 8. Dynamic Encoder-Decoder Architecture of Dynamic Attention Model

The RL agent is forced to wait for others once it arrives at x_0 .When all are in depots, the encoder with the mask is applied to the whole batch.

7 OPTIMIZATION WITH POLICY GRADIENTS

Solving combinatorial optimization problem is taken as Markov Decision Process (MDP), and the AM-D model is trained using policy gradient using Reinforcement algorithm. During training, the instances are drawn from the same distribution *X*. Given an instance X, the training objective is the tour length of solution π. Hence, based on instance X, the gradients of the parameters θ are defined as

$$\nabla_{\theta} \ J(\theta|X) = \ E_{\pi \sim p_{\theta}(.|x)}[(L(\pi|X) - b(X)) \nabla_{\theta} \log \ p_{\theta} \ (\pi|X)]$$

Where condition probability of solution is:

$$p_{\theta}(\pi|X) = \prod_{t=1}^{T} p_{\theta} \ (\pi_t|X, \pi_{1:t-1})$$

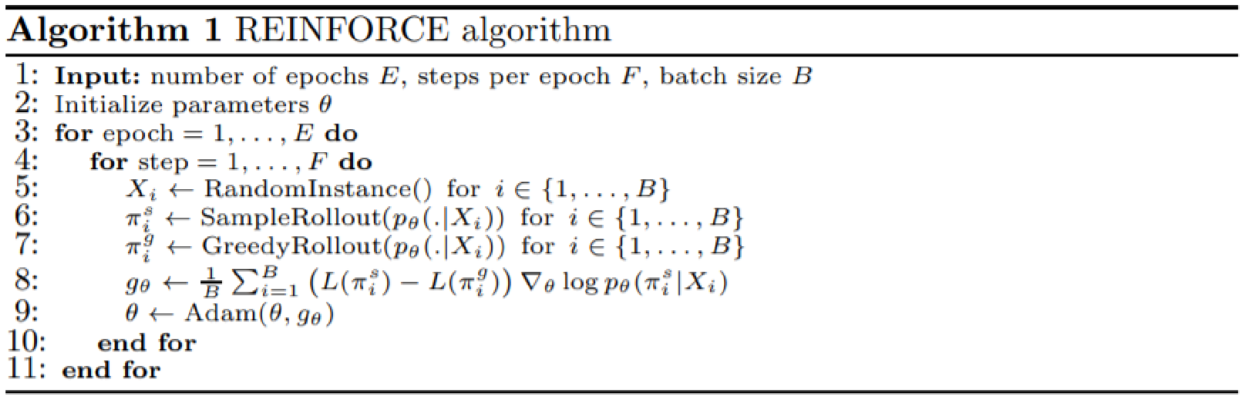
and b denotes a baseline function that does not depend on π and estimates the expected tour length to reduce the variance of the gradients. Model cost is estimated by Monte Carlo. We resort to the policy gradient method and stochastic gradient descent to optimize the parameters. The gradients of parameters θ are approximated by Monte Carlo sampling as

$$\nabla_{\theta} \ J(\theta) = \ 1/B \quad \sum_{i=1}^{B}[(L(\pi_i^s|X_i) - L(\pi_i^g|X_i)) \log \nabla_{\theta} \, p_0 \ (\pi_i^s|X_i)]$$

Where B is the batch size, π_i^s and π_i^g are the solutions of the instance X_i constructed by sample rollout and greedy rollout, respectively.

Rollout Greedy Reinforcement with baseline

The baseline is the copy of the model with fixed weights from one of the preceding epochs. A popular choice of the baseline b is to use the exponential moving average of the model cost over past epochs. The baseline is updated at the end of the epoch if the difference in the costs for the candidate model and baseline model is statistically significant (t-test). Here the baseline uses a separate dataset for this comparison and the dataset is updated after each baseline renewal.



Algorithm 1 REINFORCE algorithm

```
Input: number of epochs $E$, steps per epoch $F$, batch size $B$
Initialize parameters $\theta$
for epoch = $1, \dots, E$ do
    for step = $1, \dots, F$ do
        $X_i \leftarrow$ RandomInstance() for $i \in \{1, \dots, B\}$
        $\pi_i^s \leftarrow$ SampleRollout($p_\theta(.|X_i)$) for $i \in \{1, \dots, B\}$
        $\pi_i^g \leftarrow$ GreedyRollout($p_\theta(.|X_i)$) for $i \in \{1, \dots, B\}$
        $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i^s) - L(\pi_i^g)) \nabla_\theta \log p_\theta(\pi_i^s|X_i)$
        $\theta \leftarrow$ Adam($\theta, g_\theta$)
    end for
end for
```

Figure 9. Reinforce Model training with baseline

Estimate the model cost by Monte Carlo by generating a episode $S_1, A_1, \ldots, S_T, A_T$, following condition probability of the solution $p_\theta(\pi|X)$ in sampling mode (stochastic policy). Then loop through all the steps to get the cost of the whole episode. After this evaluate the baseline by selecting the node with the maximum probability deterministic policy (greedy mode). Estimate gradient according to policy-gradient formula and update the weights of the neural network.

7.1 Model Training

In this project, a dataset with 10,000 sample points is created, and the instances are randomly selected from the sample. For all the nodes, the location coordinate points are uniformly distributed between [0,1] and the demand is a discrete number in {1, . . . , 9} chosen uniformly at random. The demand at the depot is 0. The capacity of vehicle D = 30 for VRP with 20 customer nodes (denoted as VRP 20), D= 35 for VRP 30, D= 40 for VRP 40, D = 45 for VRP50, and the vehicle is located at the depot when t = 1. The batch size B = 128 and learning rate η = 10−4 for VRP20, VRP30 and VRP40 and η = 3*10−5 for VRP50.

Training times

The models are trained in Google Colab (Pro) which uses Telsa P100-PCIE-16GB GPU. Time to train a single epoch took varied between 10 sec to 2 mins for all the models, the optimal total number of epochs varied between 50 to 100 for VRP20 and VRP30 with a total time less than 2 hours before the cost started to increase with model overfit. In the case of VRP40, the model took around 2.5 hours with epochs varying between 120 to 150, and the VRP50 model took almost 4 hours with idle epochs varying between 150 to 200.

Hyperparameters

Parameters	VRP20	VRP30	VRP40	VRP50
Graph Size	20	30	40	50
Capacity	30	35	40	45
Training Time	50 min	95 min	2.5 hrs	4 hrs
Batch	128	128	128	128
Learning Rate	0.0001	0.0001	0.00003	0.00005
Rollout Samples	10000	10000	10000	10000
Embedding Dim	128	128	128	128
Grad Norm Clipping	1	1	1	1
Validation Batch Size	1000	1000	1000	1000
Batch Verbose	1000	1000	1000	1000
Optimal Epoch	41	95	147	291

Table 1. Hyperparameters used for different models

Due to the limitation of Google Colab computing resources the model trained only for problem instances with 50 customers. The same model can be used to train the problems with a larger instance by playing with hyperparameters, but it is time-consuming and may require multiple GPUs. Once the model is trained it will only take 0.2 milliseconds for solving each instance.

8 RESULTS AND DISCUSSIONS

Machine learning and optimization are very closely related. The main principle of machine learning works on minimizing loss function which can be costly. In many cases, Machine learning is used to reduce costs in optimization algorithms. Different from these methods, the project aims to learn heuristics from data directly without human intervention.

We compare our Reinforcement model against the savings algorithm. The savings algorithm is implemented using a vehicle routing solver from OR-Tools. Since, Google OR-Tools is one of the best available software to solve VRP problems and can tackle large problems at a higher level of generality than other solvers, instead of Gurobi Solver the Google OR-Tools is considered. The Euclidean distance is the length between two points in both cases. The optimality gap is calculated by using the formulae

Optimality gap = (SA distance- RM distance)/ SA distance *100

The models are used to solve 10 different instances and the average values are calculated. Table 2. shows the average length of both the Savings algorithm and Reinforcement model on VRP20, VRP30, VRP40, VRP50.

	Savings Algorithm	**Reinforcement Model**	**Gap**
VRP20	7.51	6.1	18.77%
VRP30	10.02	8.52	14.97%
VRP40	11.39	10.38	8.86%
VRP50	13.13	10.62	19.11%

Table 2. Avg cost comparison of ML model and Savings model

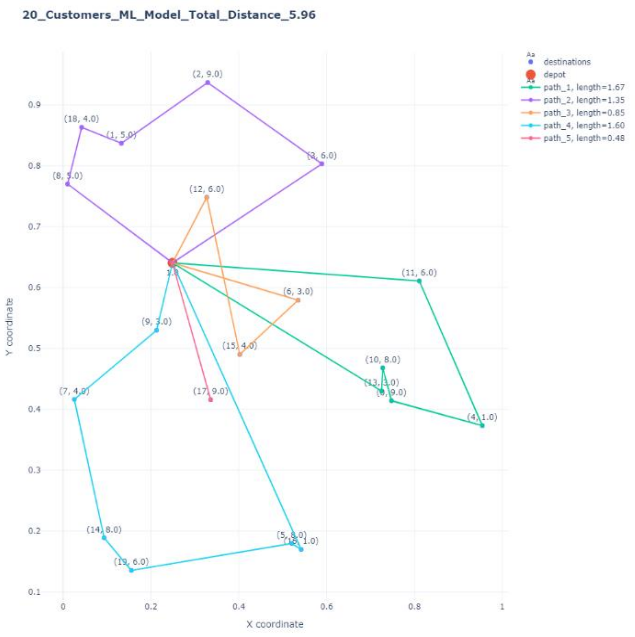
The optimal gap is significant for small and large instance problems. The results show that on average there is almost a 10-20% improvement in the cost savings when compared to Google OR Tools savings algorithm implementation, which can result in significant savings considering the vehicle is traveled multiple times every day.

8.1 Implementation

To check the performance of the model is tested. The model is tested with real-time data. UPS store located in the Vancouver area is taken as the depot. Since, customer location for deliveries can be anywhere in the region, random location coordinates for 20,50, and 100 customers are generated in the Greater Vancouver Region. The demand is uniformly generated between 1 to 9. For faster results, the location coordinates values are normalized to the range 0,1. In this problem, only one vehicle is considered. Every time the vehicle exceeds capacity it should return to the depot and then deliver to the customers again.

Smaller Instance

The below figures represent the optimal route chosen by both the models for 20 customer locations with a vehicle capacity of 30.



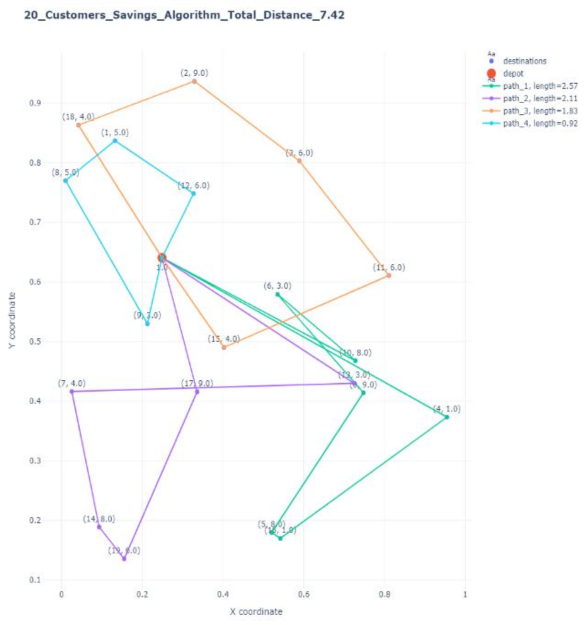
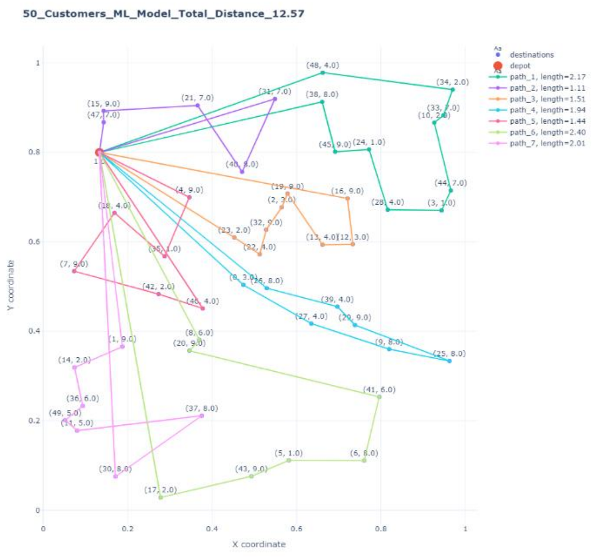


Figure 10. ML model path with a total distance of 5.96 **Figure 11.** Savings Algorithm path with a distance of 7.42

The total distance for the optimal path chosen with the ML model is almost 25% less when compared with the savings algorithm, which is very huge and can result in significant savings. It is very clear from fig 11. path 1 chosen by the savings algorithm model is not optimal.

Medium Instance

The below figures represent the optimal route chosen by both the models for 50 customer locations with a vehicle capacity of 45.



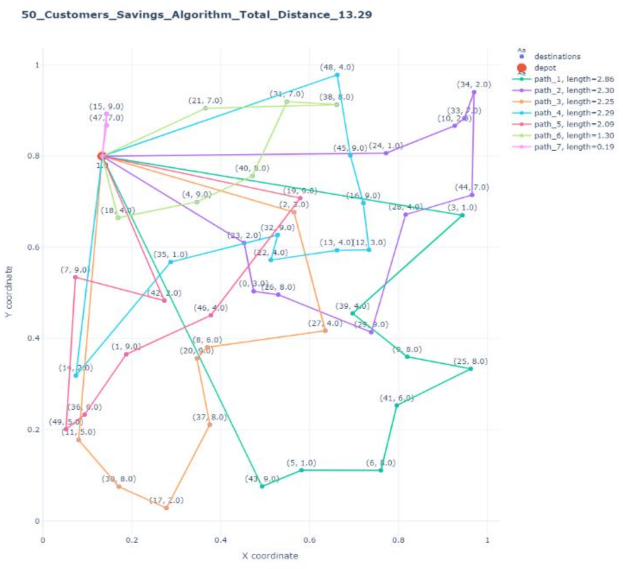
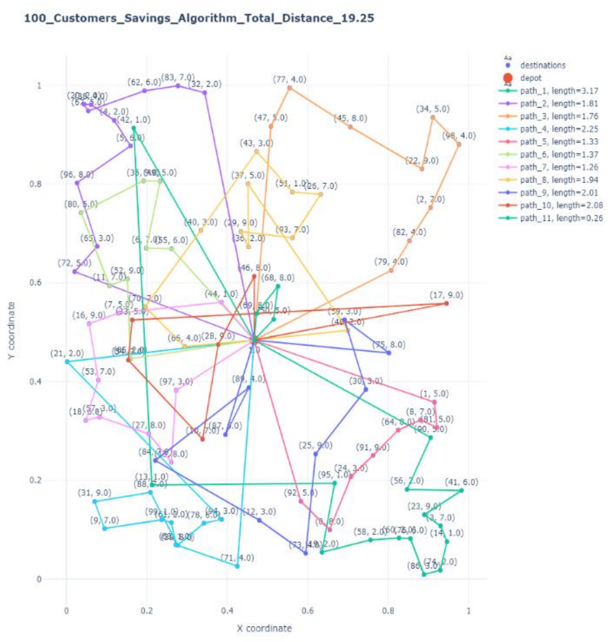


Figure 12. ML model path with a distance of 12.57 **Figure 13.** Savings Algorithm path with a distance of 13.29

Here, for 50 customer locations, the total distance for the optimal path chosen with the ML model is around 6% less when compared with the savings algorithm.

Large Instance

The below figures represent the optimal route chosen by both the models for 100 customer locations with a vehicle capacity of 50.



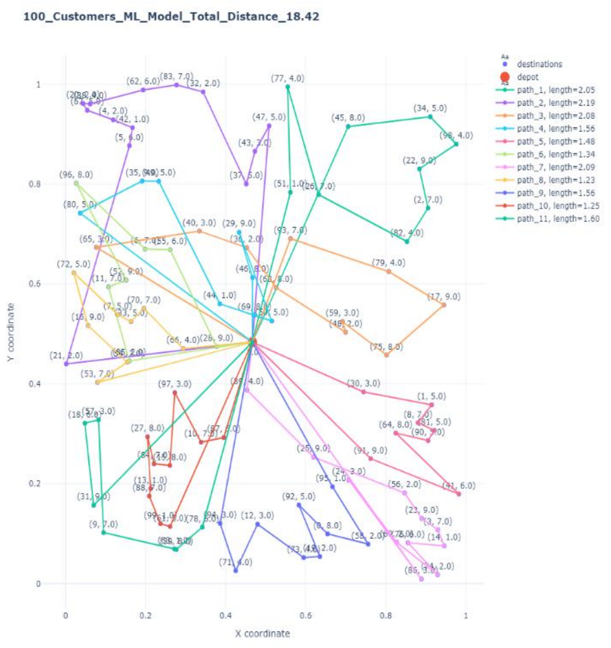


Figure 14. ML model path with a total distance of 18.42**Figure 15.** Savings Algorithm path with a distance of 19.25

Here, for 100 customer locations, the total distance for the optimal path chosen with the ML model is around 5% less when compared with the savings algorithm.

In case of medium and large problems, the total cost savings using the Reinforcement model is around 5-10% which does not seem to be a huge difference when compared with the savings algorithm, but since the vehicles are traveled multiple times every day, even 5% less travel distance can result in saving so much travel cost in long run.

8.2 Experimentation

To verify whether the model can be generalized without training individually for a specific set of customers every time. All the models are tested against other different model which are trained to solve other instances to compare the performance of smaller instance models on medium and large instance problems and vice versa.

Table 3. shows the total length of both the models when used to solve different instances.

Total No. of Customers	Savings Algorithm	VRP20 Model	Gap	VRP30 Model	Gap	VRP40 Model	Gap	VRP50 Model	Gap
20 Customers	7.25	**6.56**	9.51%	7.07	2.48%	6.91	4.68%	6.89	4.96%
30 Customers	9.92	9.32	6.04%	**9.1**	8.26%	9.22	7.05%	9.87	0.5%
40 Customers	11.37	10.94	3.78%	11.37	0%	**10.59**	6.86%	11.02	3.07%
50 Customers	14	12.81	8.5%	11.18	20.14%	10.64	24%	**10.27**	26.6%
100 Customers	19.33	19.9	-2.94%	19.68	-1.81%	18.77	2.89%	**18.47**	4.44%

Table 3. Comparison of the ML model and Savings model for different instances

The results show that the generalization performance of the model is very good. It can be seen that the reinforcement model that is already trained to solve the specific set of customers performed significantly better when compared with the saving algorithm. Also, the other models which are not trained specifically to solve a particular customer's instance performed slightly better or almost equal to the savings algorithm.

Even though the reinforcement model is not trained to solve 100 customers, when tested with all the trained models, the VRP50 model performed 5% better than the Savings Algorithm, but the VRP 20 and VRP 30 models performance were slightly lower than the savings algorithm.

To conclude, the models trained to solve smaller instances can be used to solve small and medium problems, but when applied to the larger instance problems the performance is a bit less when compared to google OR Tools. In the case of models trained to solve large instances, the optimality gap is very less when compared with the optimal route of the saving algorithm method. Nevertheless, reinforcement models performed better than the savings algorithm in most of the instances.

9 CONCLUSION

In this project, a dynamic attention model with a dynamic encoder-decoder framework is presented to solve Capacitated VRP, to learn and design heuristics based on data without human intervention. It is proven that the heuristics designed by the machine learnings are better than handcrafted heuristics. The results show that the AM-D model performed significantly better for solving VRP. Also, the AM-D model showed good generalization performance when tested with different problem scales. The proposed model is not limited to VRP and has the potential to be used in real-world problems. The proposed AM-D model with some modifications can be used to solve other complex VRP variants like multi depot CVRP problems and VRP with time windows and can be extended to solve other combinatorial optimization problems such as Job scheduling, Assignment problems which will open endless possibilities.

Using the Reinforcement model showed around 10-20% improvement when compared with the Savings Algorithm. This model is quite fascinating because of the generalization and the time taken to find the optimal solution. Once models are trained to solve small, medium, and large instances they can be used to find optimal solutions for a huge variety of problems with different instances.

REFERENCES

[1] Peng B., Wang J., Zhang Z. (2020) A Deep Reinforcement Learning Algorithm Using Dynamic Attention Model for Vehicle Routing Problems. In: Artificial Intelligence Algorithms and Applications. ISICA 2019. Communications in Computer and Information Science, vol 1205. Springer, Singapore.

[2] Sreeram V. B.(1990), "A theoretical investigation into the performance of the Hopfield model", IEEE Transactions on Neural Networks, Vol. 204 No. 215.

[3] Bello, I. (2016), Neural combinatorial optimization with reinforcement learning. Retrieved from https://arxiv.org/1611.09940.

[4] Gao, L., Chen, M., Chen, Q., Luo, G., Zhu, N., & Liu, Z. (2020, February 20). Learn to Design the Heuristics for Vehicle Routing Problem. Retrieved from https://arxiv.org/abs/2002.08539

[5] Kool, W., van Hoof, H., Welling, M. (2019) Attention, learn to solve routing problems! International Conference on Learning Representations.

[6] Nazari, M. R., Afshin, O., Lawrence, V. S., Martin, T.(2018). Reinforcement learning for solving the vehicle routing problem. NeurIPS.

[7] Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M. (2018). Learning heuristics for the TSP by policy gradient. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer 170–181.

[8] Jonathan, n.d. RL policy Gradients Explained [Online], Available from https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146 [Accessed: 3rd December 2020]

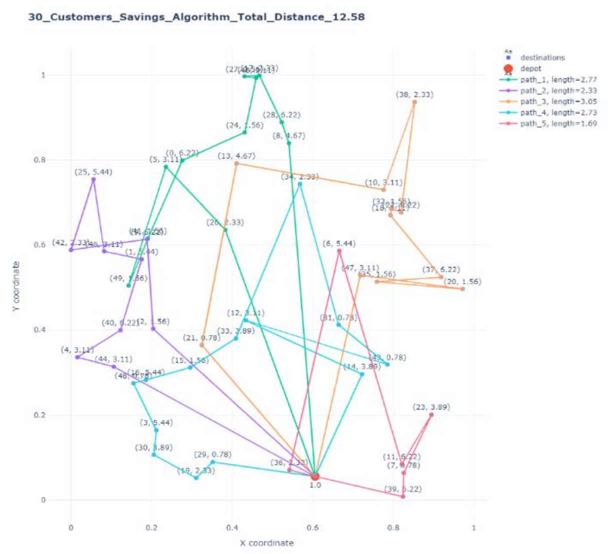
[9] Savings Algorithm [Online], Available from https://neo.lcc.uma.es/vrp/solution-methods/heuristics/savings-algorithms/ [Accessed: 3rd December 2020]

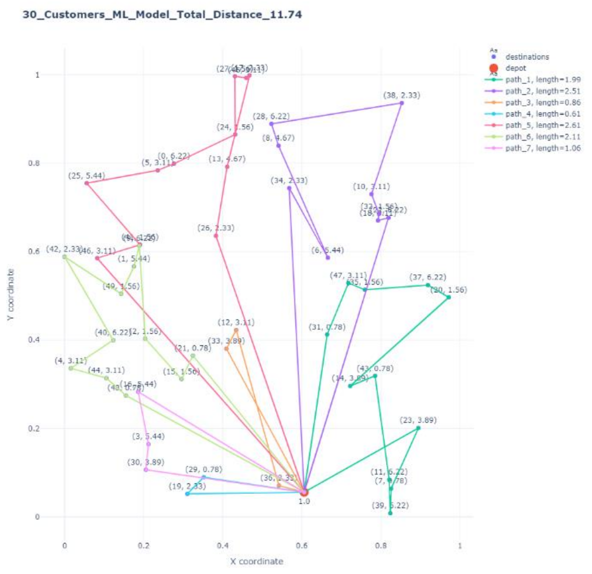
APPENDIX

The detailed code for this project and the trained models can be found in my GitHub repo here: https://github.com/JaswanthBadvelu/Reinforcement-Learning-CVRP

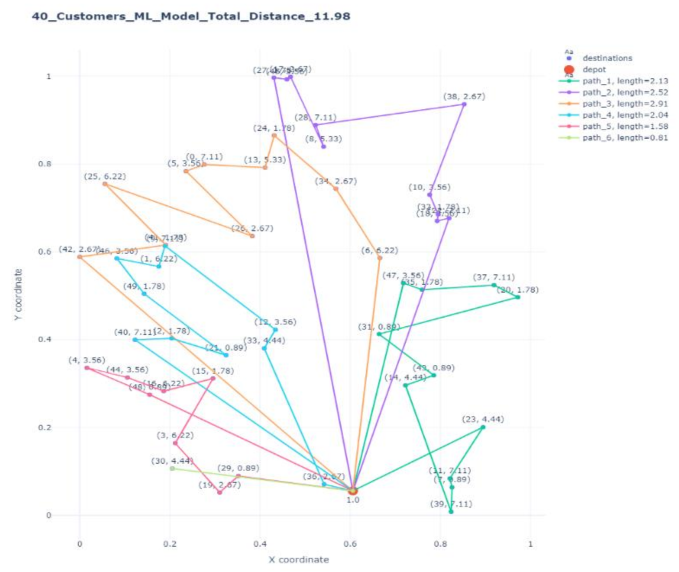
Samples of Output

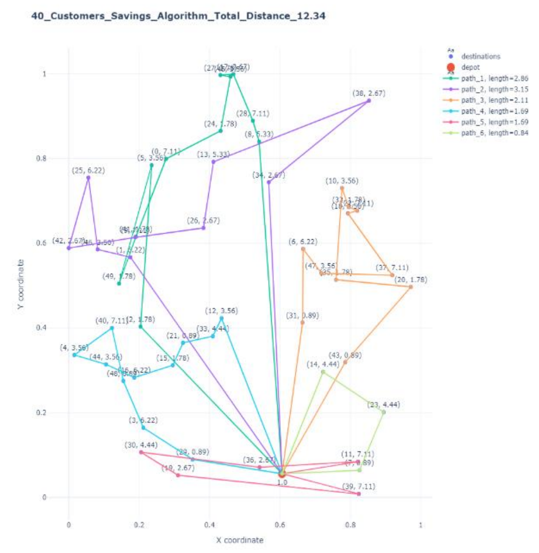
For 30 customers





For 40 customers


40_Customers_ML_Model_Total_Distance_11.98
destinations
depot
path_1, length=2.13
path_2, length=2.52
path_3, length=2.91
path_4, length=2.04
path_5, length=1.58
path_6, length=0.81
X coordinate
Y coordinate


40_Customers_Savings_Algorithm_Total_Distance_12.34
destinations
depot
path_1, length=2.86
path_2, length=3.15
path_3, length=2.11
path_4, length=1.69
path_5, length=1.69
path_6, length=0.84
X coordinate
Y coordinate