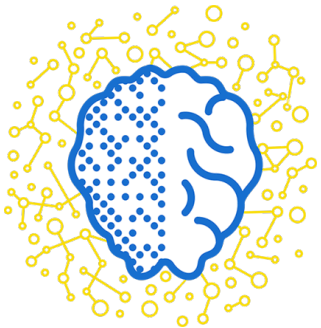
Dynamic Attention Model for Vehicle Routing Problems

Dmitry Eremeev, Alexey Pustynnikov

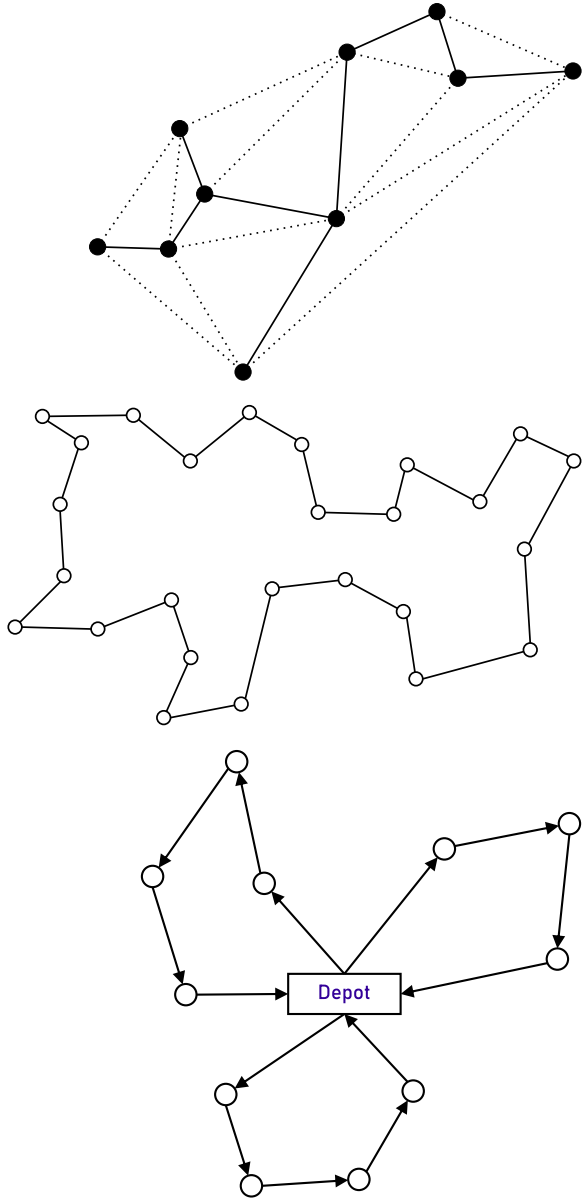
Advanced Topics in Deep Reinforcement learning

DeepPavlov.ai



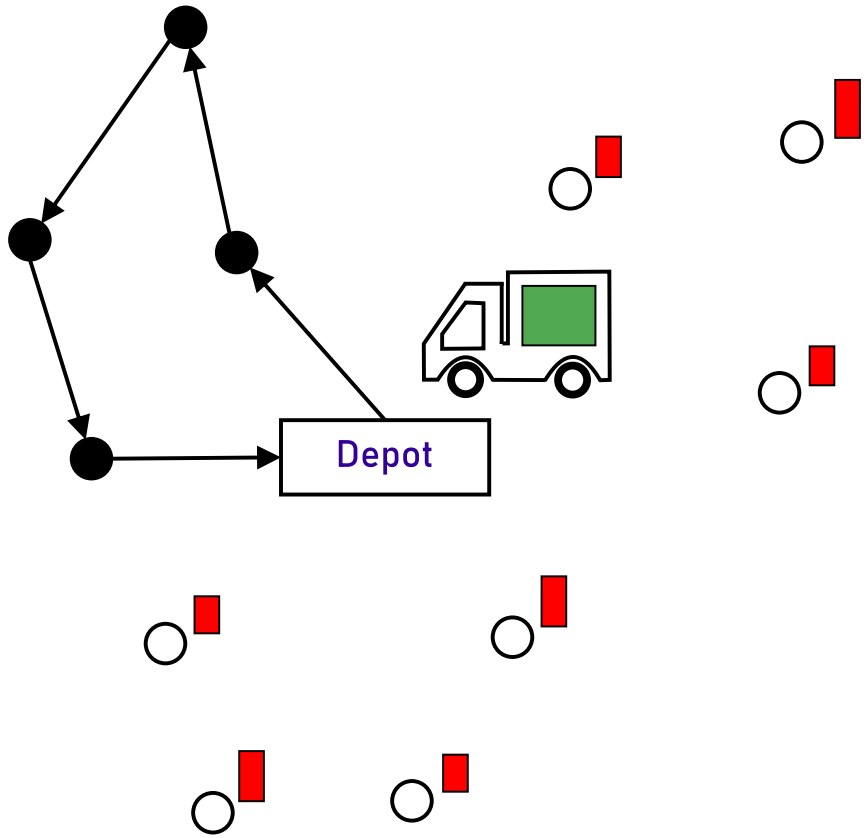
Combinatorial Optimization

- Minimum spanning tree
- Travelling Salesman Problem
- Vehicle Routing Problem



Capacitated Vehicle Routing Problem

Particular case of VRP: 1 vehicle with a limited carrying capacity.



CVRP: Graph Formulation

- Complete graph

$$X = (V, E),$$

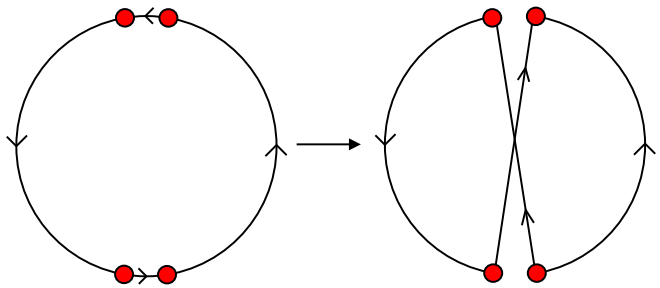
with set of nodes

$$V = \{x_0 \equiv D, x_1, x_2, \ldots, x_n\}.$$

- Each node is associated with a demand d_i, $d_i < C$, $d_D = 0$.
- Each edge is associated with a cost c_{ij} (L^2-norm).
- A vehicle with capacity C is moving along graph starting from depot node D. Every non-depot node x_k can be visited only once. It is allowed to return to D arbitrary many times.
- Goal: find a path $\pi = \{\pi_1, \ldots, \pi_T\}$, $\pi_t \in V$ that minimizes **total cost**.

Heuristics and approximate solution

- VRP is an NP-hard problem (Lenstra and Rinnooy Kan, 1981). Exact algorithms are only efficient for small problem instances.
- The number of near-optimal algorithms are introduced in academic literature. There are multiple professional tools for solving various VRP problems (ex. Google OR-Tools).
- LKH algorithm: highly effective implementation (Helsgaun-2000, ...) of Lin–Kernighan (1973) heuristic.



Flip operation

1) Generate random initial solution.
2) Construct the alternating path by a sequence of flip operations.
3) These flips replace k edges x_i in current tour (circle) by edges y_i which are NOT in current tour. Keep increasing k as long as cost decreasing.

Deep Learning for VRP

ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS!

Wouter Kool
University of Amsterdam
ORTEC

Herke van Hoof
University of Amsterdam
h.c.vanhoof@uva.nl

Max Welling
University of Amsterdam
CIFAR

arXiv:1803.08475 [stat.ML]

A Deep Reinforcement Learning Algorithm Using Dynamic Attention Model for Vehicle Routing Problems

Bo Peng, Jiahai Wang(✉), and Zizhen Zhang

Department of Computer Science, Sun Yat-sen University, Guangzhou, China

arXiv:2002.03282 [cs.LG]

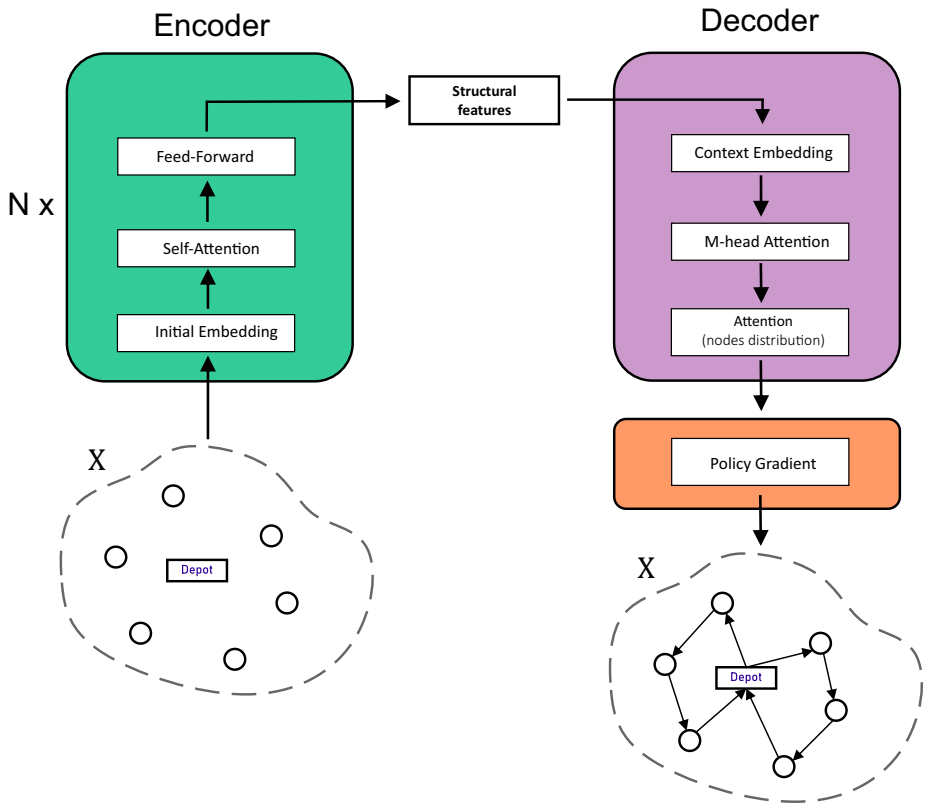
Share common idea: utilize Reinforcement Learning (RL) to generate solutions

- Make use of Graph Attention Networks (GAT) to obtain graph embeddings.
- Train RL agent that can learn heuristics and provide suboptimal solutions.

Differ in their approaches to generating embeddings

Encoder-Decoder architecture

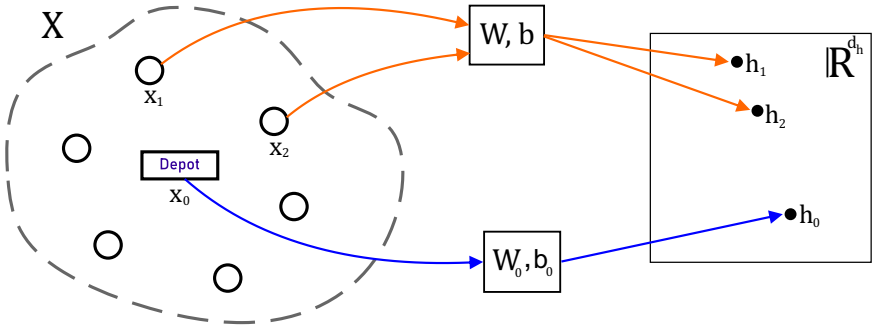
VRP problem as a sequential decision making problem



- At each decoding construction step, one node is selected and appended to the current solution.
- Probability of solution for path $\pi = \{\pi_1, \ldots, \pi_T\}$ and graph instance X:

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

Encoder in detail: Initial Embedding



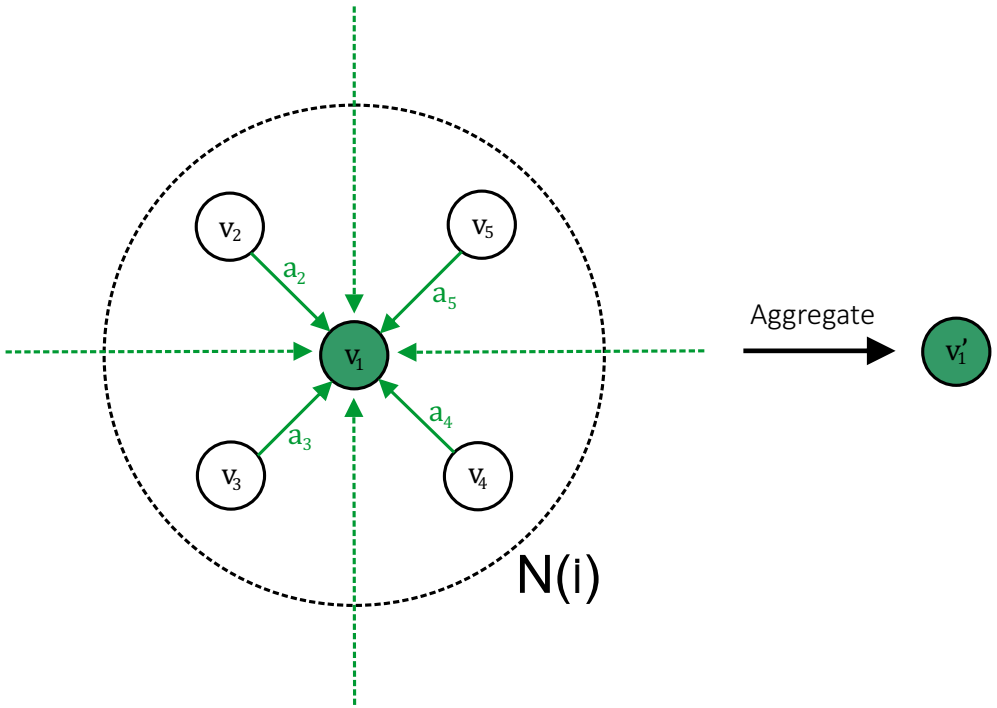
$$h^{(0)} : \mathbb{R}^3 \hookrightarrow \mathbb{R}^{d_h}, d_h = 128$$

- For each node linearly project $x_i = (s_{1i}, s_{2i}, \text{demand}_i) \in \mathbb{R}^3$ to $\mathbb{R}^{128}$
- Learnable parameters $W \in \mathbb{R}^{3 \times d_h}$, $b \in \mathbb{R}^{d_h}$. Also separate parameters W_0, b_0 are used for the depot:

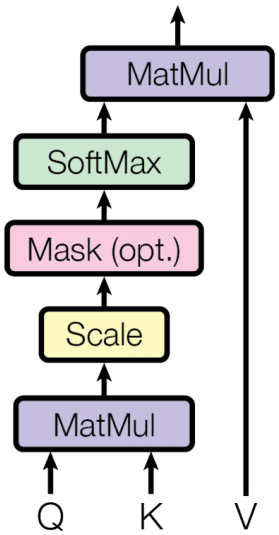
$$h_i^{(0)} = \begin{cases} W \cdot x_i + b & i \neq 0 \\ W_0 \cdot x_i + b_0 & i = 0 \end{cases}$$

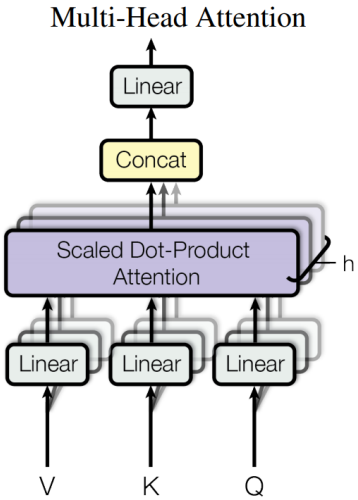
Encoder in detail: Message Passing

- Use neural message passing via self-attention (Velickovic et. al., 2018)
- Embedding (hidden state) for each node $\mathbf{v}_i$ is obtained by a weighted sum of features of all nodes $\mathbf{v}_{j \in N(i)}$ in some neighbourhood $N(i)$.
- Weights a_i are calculated by attention mechanism, representing the importance (similarity) of each neighbour for a specific node.



Encoder in detail: Multi-Head Attention (Vaswani et al., 2017)





- Linearly project initial node embeddings $H \in \mathbb{R}^{batch \times n \times d_h}$ (query, key, value):

$$Q = HW^Q, K = HW^K, V = HW^V,$$

$$W^Q, W^K, W^V \in \mathbb{R}^{d_h \times d}.$$

- Split into M heads and compute compatibility matrix $A \in \mathbb{R}^{batch \times M \times n \times n}$ for graph nodes:

$$A = \text{softmax}\left(\frac{U}{\sqrt{d/M}}\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d/M}}\right).$$

- Compute attention messages for each head:

$$H' = AV \in \mathbb{R}^{batch \times M \times n \times d/M}.$$

- Concatenate heads and project out with $W^O \in \mathbb{R}^{d \times d}$:

$$MHA = \text{Concat}\left(H'_1, \ldots, H'_M\right) W^O \in \mathbb{R}^{batch \times n \times d}.$$

Encoder in detail: Feed-Forward Sublayer

For each node i apply fully connected feed-forward (FF) network with skip-connections, $\ell \in \{1, \ldots, N\}$:

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

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

$$h_i^{(\ell)} = \tanh(\hat{h}_i^{(\ell)} + \mathrm{FF}(\hat{h}_i^{(\ell)})).$$

Finally, after N layers we get final node embeddings:

$$h_i^N = \mathrm{ENCODE}_i^N\left(h_0^0, \ldots, h_n^0\right).$$

Decoder in detail: Context Vector

At each construction step $t \in 1, \ldots, T$ concatenate mean graph embedding over all nodes, embedding of the previously selected node and remaining capacity of the vehicle:

$$\hat{\mathbf{h}}_c = \begin{cases} \left[\bar{h}_t; h_0^N; D_t\right] & t = 1 \\ \\ \left[\bar{h}_t; h_{\pi_{t-\mathbf{1}}}^N; D_t\right] & t > 1 \end{cases}$$

Decoder in detail: Attention for output probabilities

Policy is governed by two sequential attention layers in decoder.

- Query vector from context vector: $\mathbf{q} = W^Q \hat{\mathbf{h}}_c$. Keys and Values from embeddings of nodes.
- Add mask to attention: mask nodes that have already been visited or have too much demand.

1. M-head attention layer

$$\mathbf{q} = W^Q \hat{\mathbf{h}}_c, \mathbf{k}_i = W^K \mathbf{h}_i, \mathbf{v}_i = W^V \mathbf{h}_i,$$

$$u_j = \begin{cases} \mathbf{q} \cdot \mathbf{k}_j^T & d_j \leq D_t, x_j \notin \pi_{1:t-1} \\ -\infty & \text{otherwise} \end{cases}$$

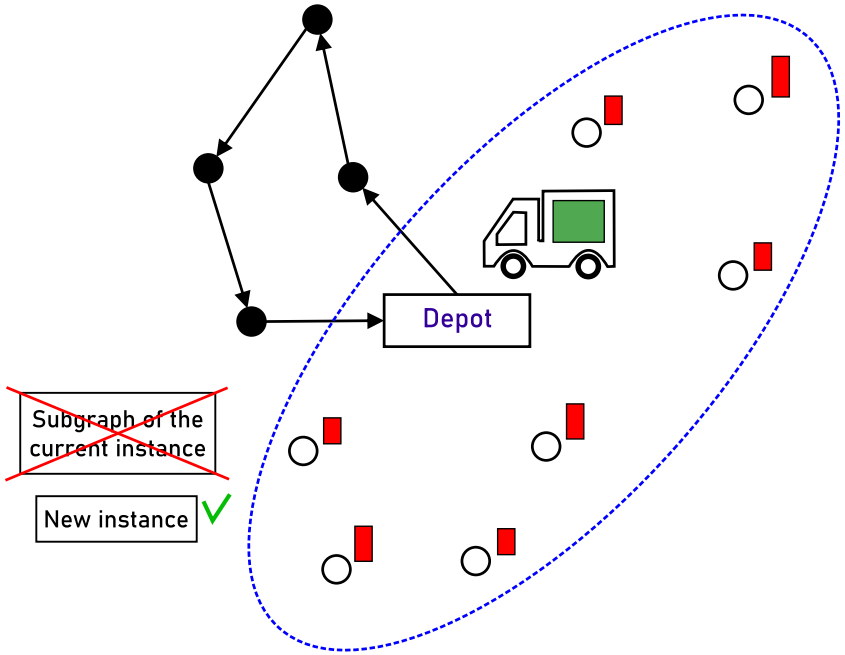
2. Single-head attention layer (only compatibility) for probabilities:

$$\mathbf{k}_{\mathbf{tanh}\,i} = W^{K_{tanh}} \mathbf{h}_i,$$

$$u_j = \begin{cases} C \cdot \tanh(\mathbf{q} \cdot \mathbf{k}_{\mathbf{tanh}_j}^T) & d_j \leq D_t, x_j \notin \pi_{1:t-1} \\ -\infty & \text{otherwise} \end{cases}$$

$$\boxed{p_\theta(\pi_t | X, \pi_{1:t-1}) = \frac{e^{u_j}}{\sum_{j'} e^{u_{j'}}}}$$

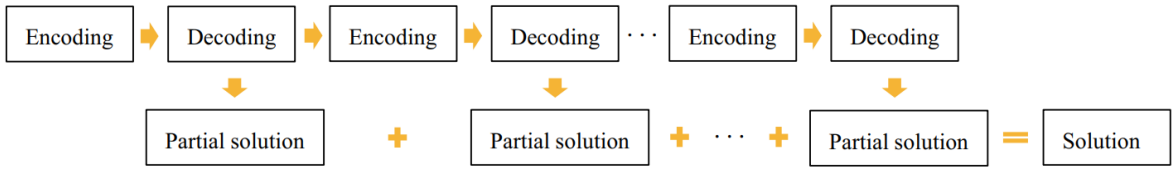
Dynamic Attention Model (AM-D)



- After vehicle returns to depot, the remaining nodes could be considered as a new (smaller) instance (graph) to be solved.
- Idea: update embedding of the remaining nodes using encoder when agent arrives back to depot.

$$h_i^t = \begin{cases} \text{ENCODE}_i^N(h_0^0, \ldots, h_N^0) & \pi_{t-1} = x_0 \\ h_i^{t-1} & \text{otherwise} \end{cases}$$

Dynamic Encoder-Decoder



Implementation:

- Force RL agent to wait for others once it arrives to x_0.
- When all are in depots, apply encoder with mask to the whole batch.
- Typical solution will be of the form

```
[17.,  3.,  4.,  7.,  2., 16.,  0.,  0.,  0.,  0., 15., 20.,  5.,
  0.,  0.,  0.,  0., 11., 12., 13., 10.,  9.,  0.,  0.,  0., 19.,
  6.,  8., 14.,  1., 18.,  0.],
```

We train model using policy gradient:

Theorem (Policy Gradient)

Gradient of expected cost for episode:

$$\nabla_\theta J(\theta) \sim \mathbb{E}_p \left[(L^p(X, \pi) - b(X)) \nabla_\theta \log(p_\theta(\pi|X)) \right],$$

where conditional probability of solution is:

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

and b is baseline.

Model Training: baseline b

- Baseline is a copy of a model with fixed weights from one of the preceding epochs.
- Use warm-up for early epochs: mix exponential moving average (controlled by $\beta = \text{const}$) of model cost over past epochs with baseline model. Warm-up is controlled by $\alpha \in [0; 1]$.
- Update baseline at the end of epoch if the difference in costs for candidate model and baseline is statistically-significant (t-test).
- Baseline uses separate dataset for this comparison. This dataset is updated after each baseline renewal.

Model Training: REINFORCE with baseline

- Estimate model cost by Monte Carlo: generate an episode $S_1, A_1, \ldots, S_T, A_T$, following $p_\theta(\cdot|\cdot)$ in sampling mode (stochastic policy). Then loop through all steps to get cost of the whole episode.
- Evaluate baseline in greedy mode (select the node with maximum probability - deterministic policy).
- Estimate gradient according to policy-gradient formula and update weights of the neural network.

Input: number of epochs E, steps per epoch F, batch size B
Initialize parameters θ

for *epoch* $= 1$ to E **do**
  for *step* $= 1$ to F **do**
    $X_i =$ RandomInstance() for $i \in 1, \ldots, B$
    $\pi_i^s =$ SampleRollout$(p_\theta(.|X_i))$ for $i \in 1, \ldots, B$
    $\pi_i^g =$ GreedyRollout$(p_\theta(.|X_i))$ for $i \in 1, \ldots, B$
    $\nabla \mathcal{L} = \frac{1}{B} \sum_{i=1}^{B} \left(L(\pi_i^s) - L(\pi_i^g) \right) \nabla_\theta \log p_\theta(\pi_i^s|X_i)$
    $\theta =$ Adam$(\theta, \nabla \mathcal{L})$
  end for
end for

- Generate new training dataset (1 280 000 random graph instances) at the beginning of each epoch.
- Create and save validation dataset (10 000 graph instances) with fixed seed before first epoch. At the end of each epoch validate model in greedy mode.

Common Parameters

Parameter	Value
Embedding dimension	128
Rollout Samples	10000
Number of warmup epochs	1
Gradient norm clipping	1.0
Validation batch size	1000
Validation set size	10000
Number of heads in MHA	8
Tanh Clipping (C)	10
FF Hidden Layer neurons	512
Warmup Exponential Beta	0.8

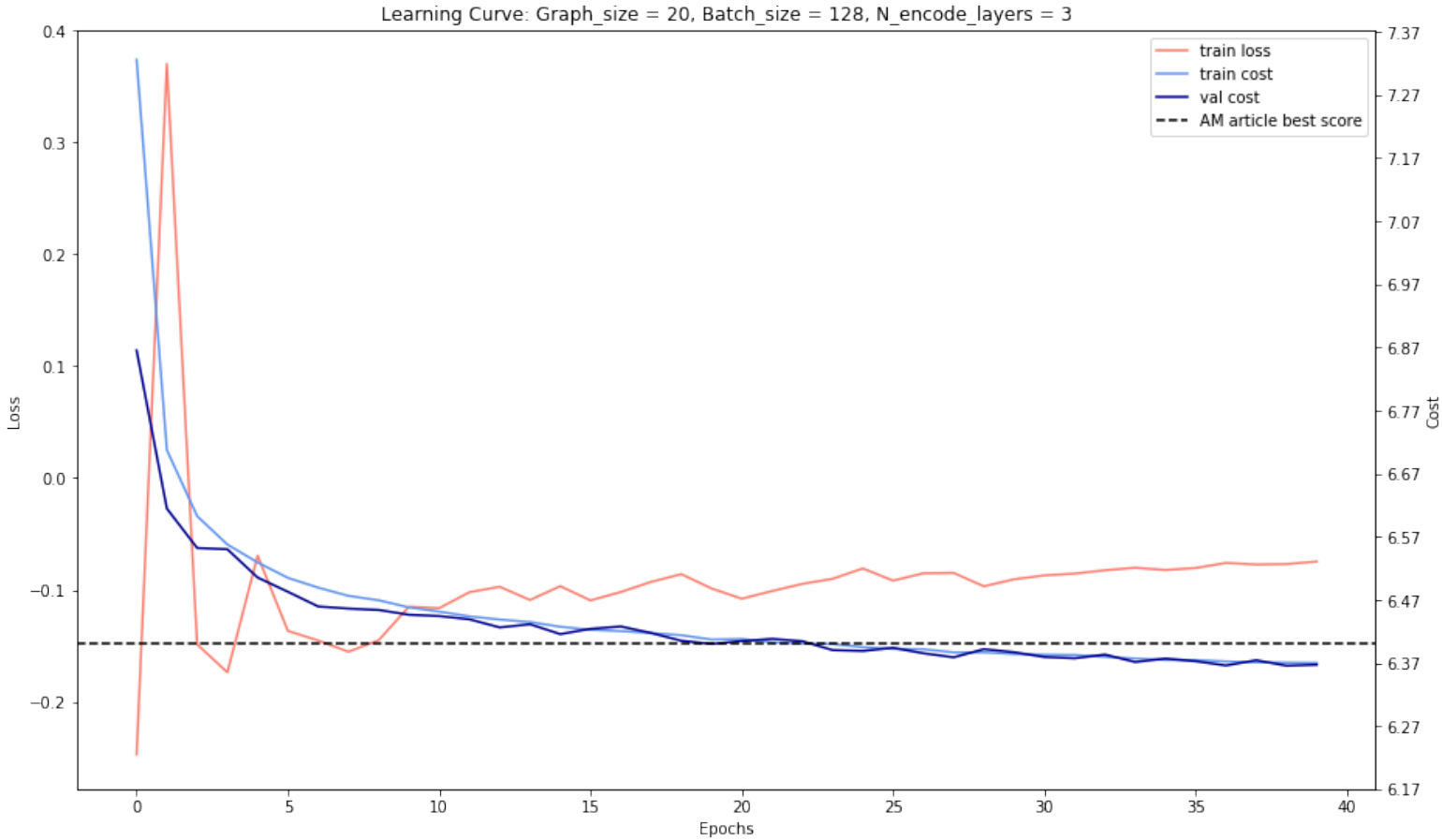
Results and Experiments

- Our implementation is based on TensorFlow 2.1 TensorFlow
- https://github.com/d-eremeev/ADM-VRP
- We looked at model behaviour for graph sizes (number of nodes) of 20/50, varying batch size and learning rate.
- For graph size 20 we tested different number of encoding layers: 2, 3.
- To compare our result with Lin–Kernighan heuristic, we used LKH-3 binaries from http://akira.ruc.dk/~keld/research/LKH-3/. We evaluated it on the same validation sets as for AM-D. Quality and inference time differ drastically.

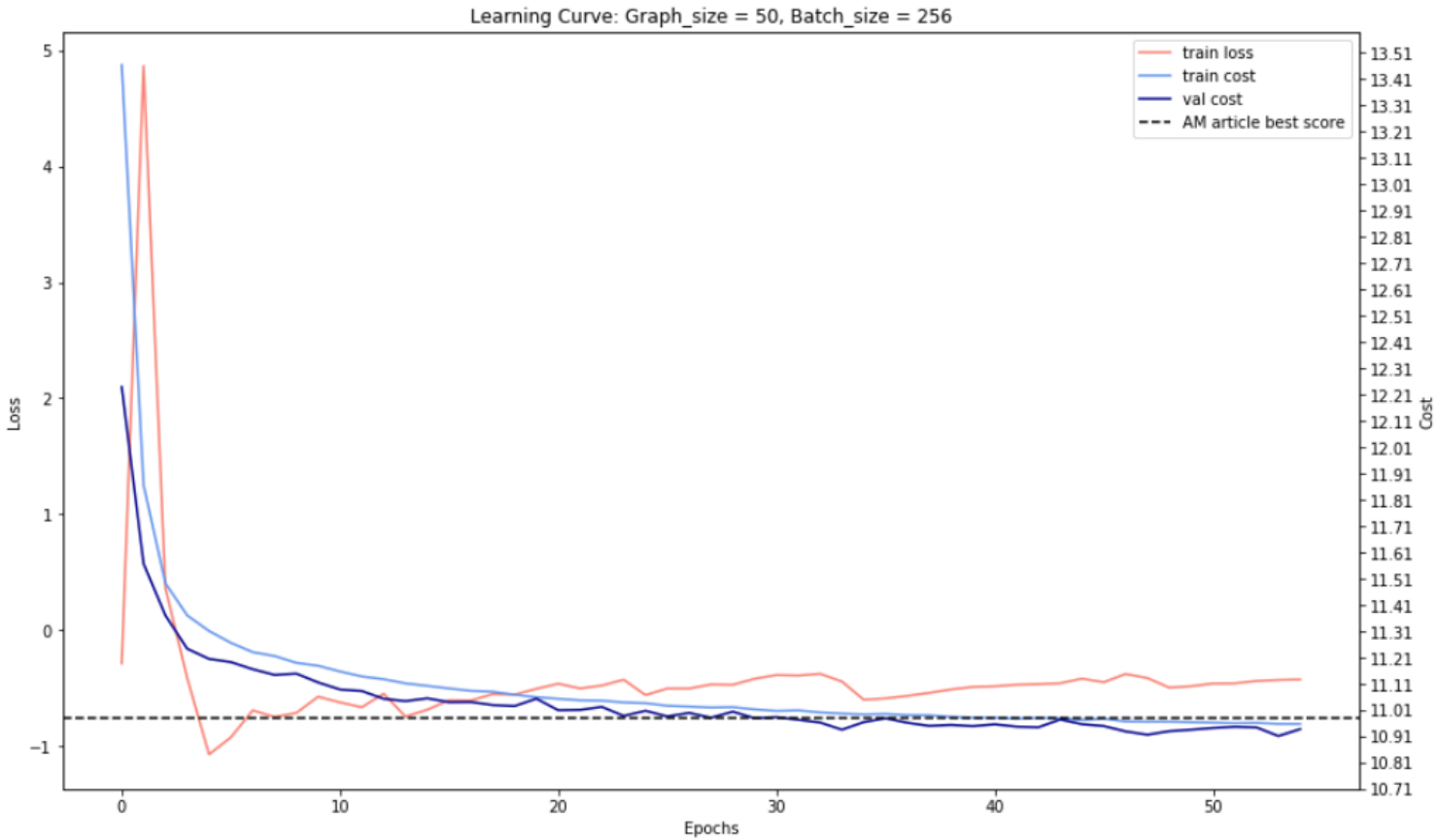
Experiments

Experiment ID	Number of nodes	Batch size	Number of epochs	Number of MHA-layers	Learning Rate	Test cost	Test cost (article AM)	Test cost (article D-AM)	Test cost baseline (LKH-3)	Mean Inference time (s)	Mean Inference time (s) (LKH-3)
1	20	1024	100	2	0.0001	6.35	6.40	6.28	6.13	0.19	4.98
2	20	128	40	2	0.0001	6.38	6.40	6.28	6.13	0.2	4.98
3	20	128	40	3	0.0001	6.36	6.40	6.28	6.13	0.2	4.98
4	50	256	55	2	0.0003	10.91	10.98	10.78	10.08	0.44	18.78

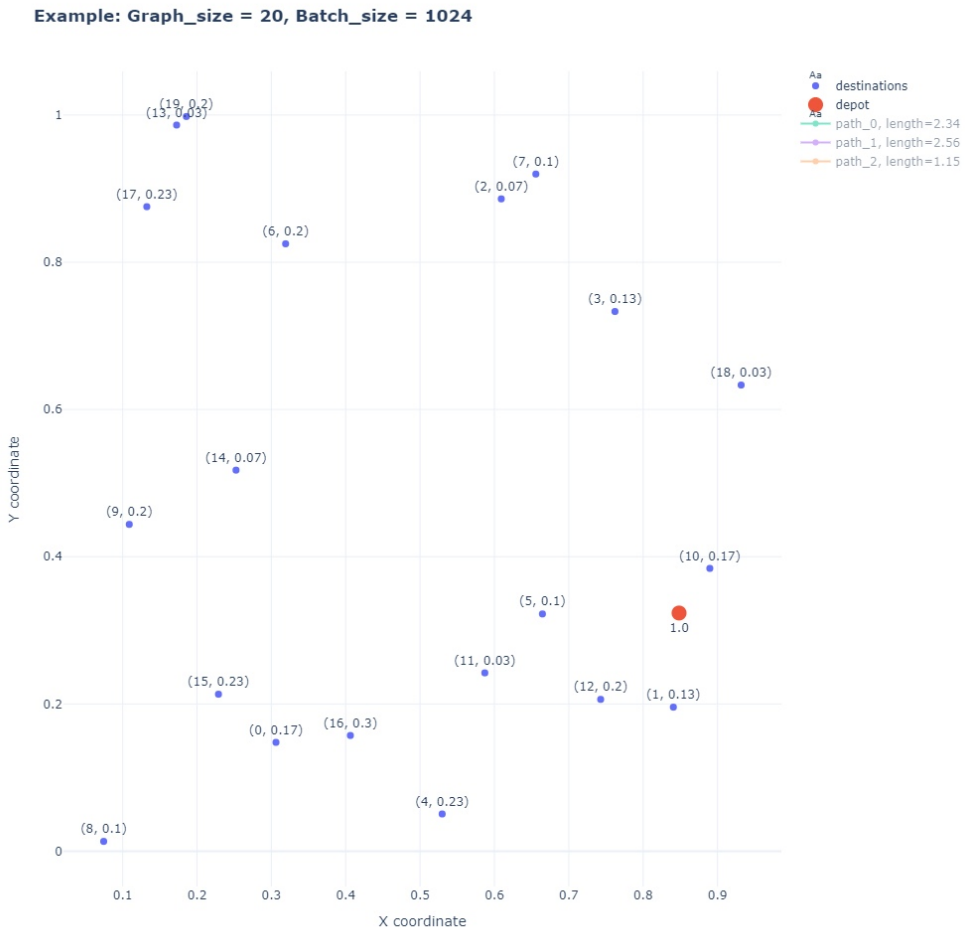
Learning Curve: VRP 20



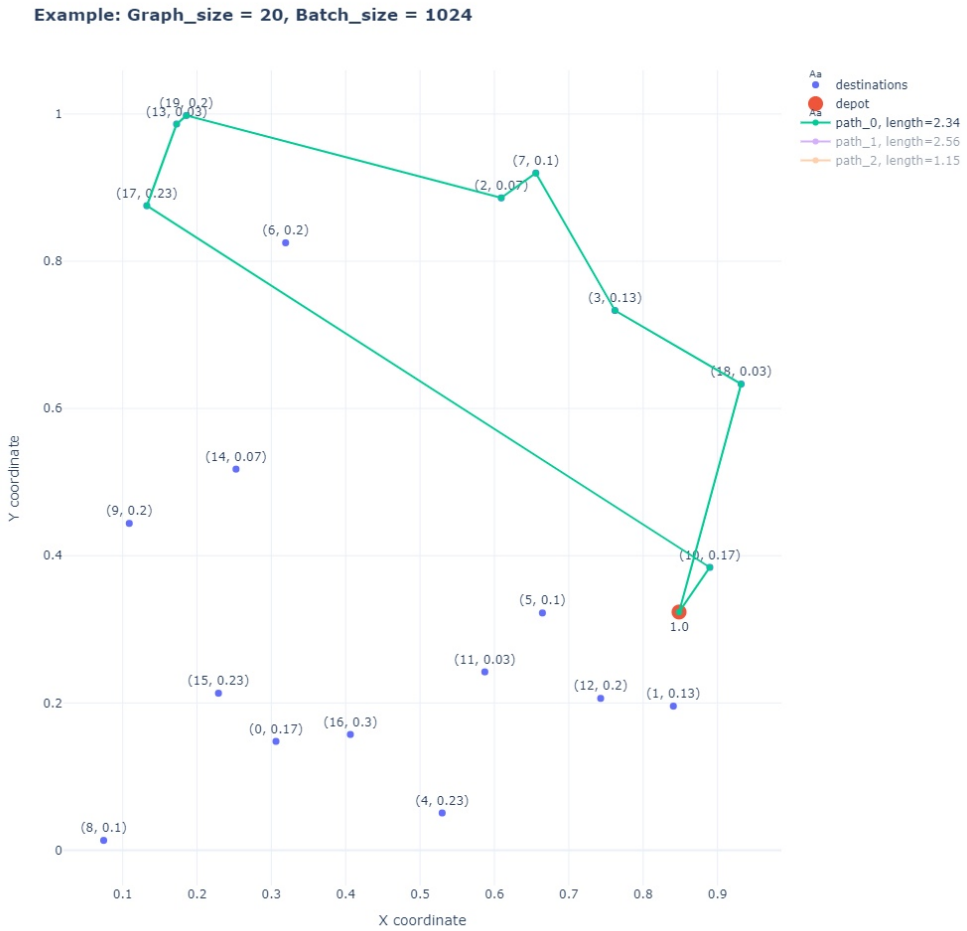
Learning Curve: VRP 50



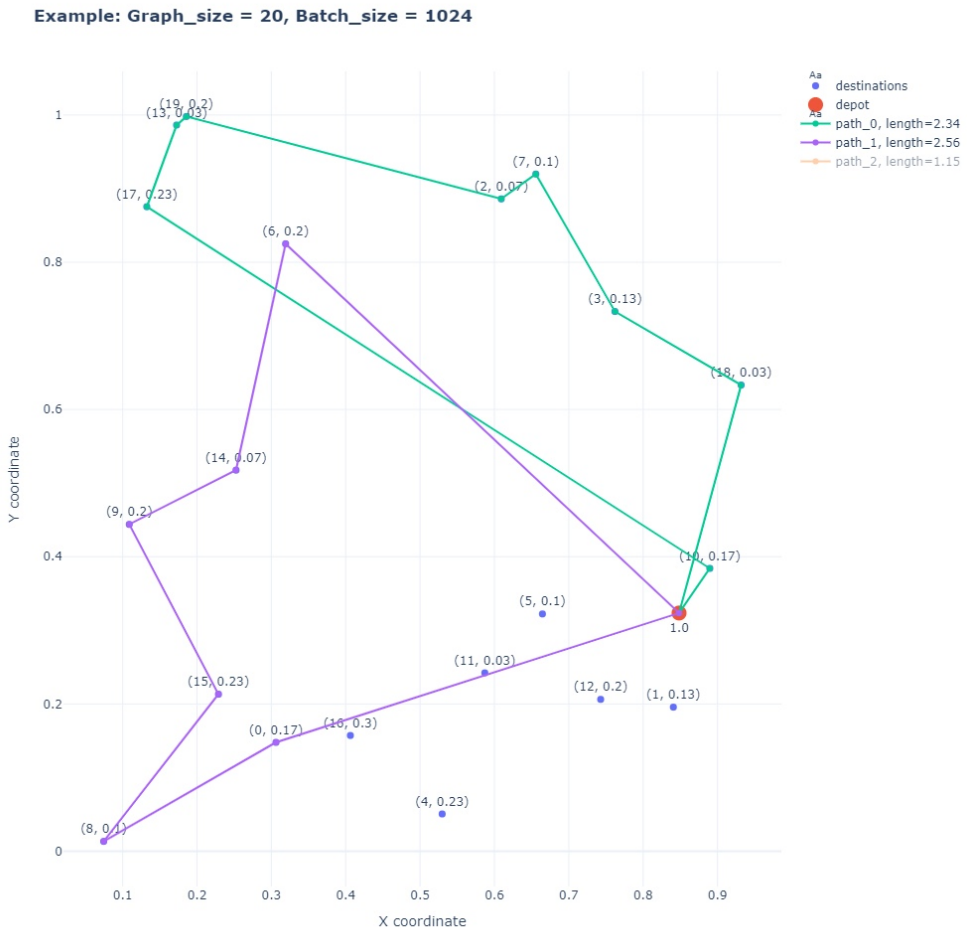
Solving VRP20



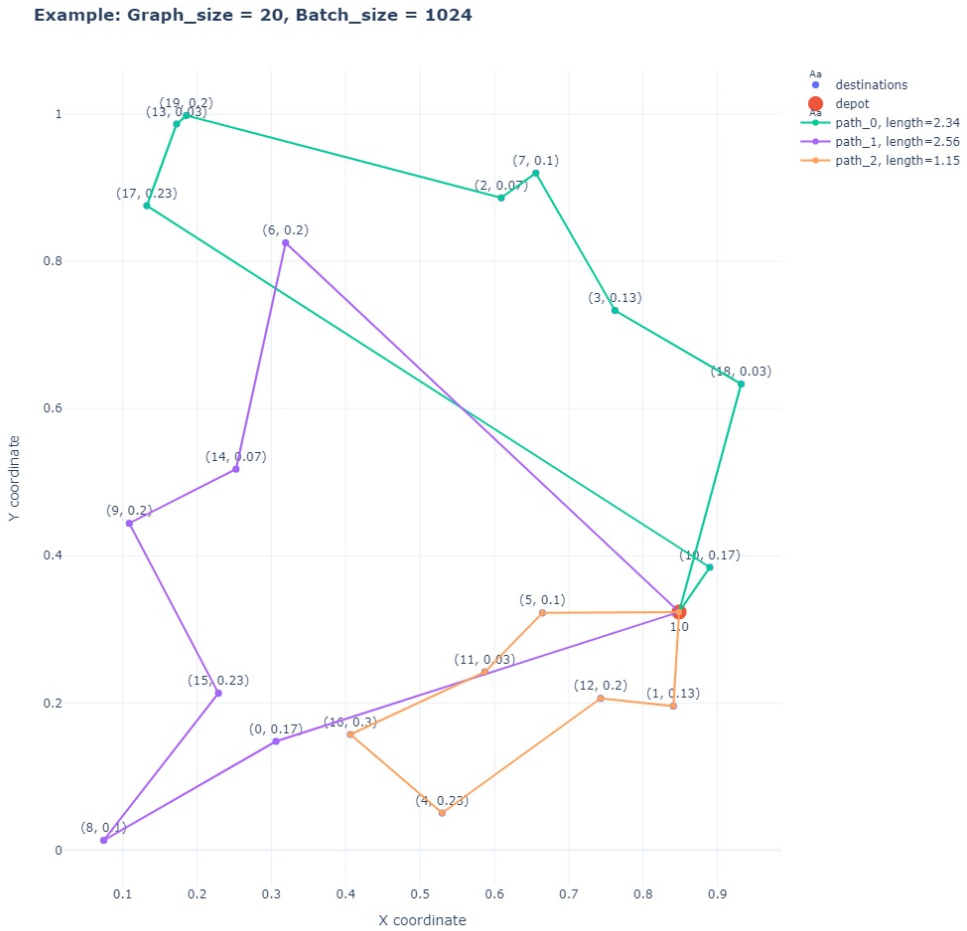
Solving VRP20



Solving VRP20



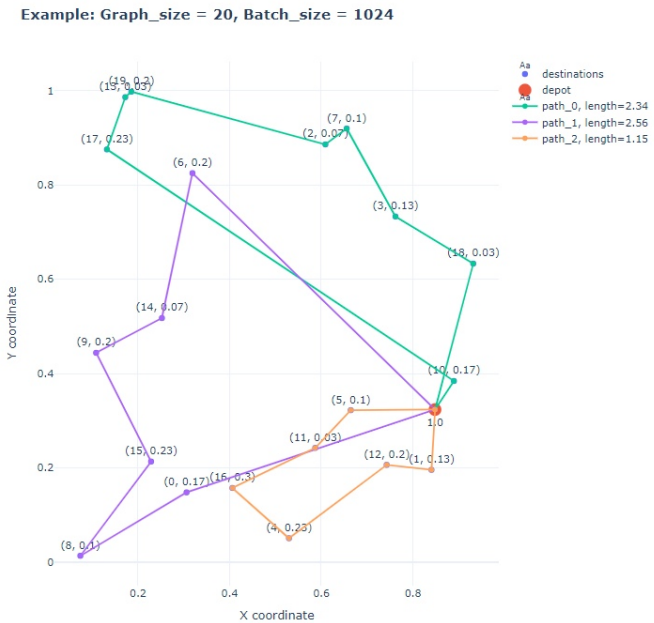
Solving VRP20



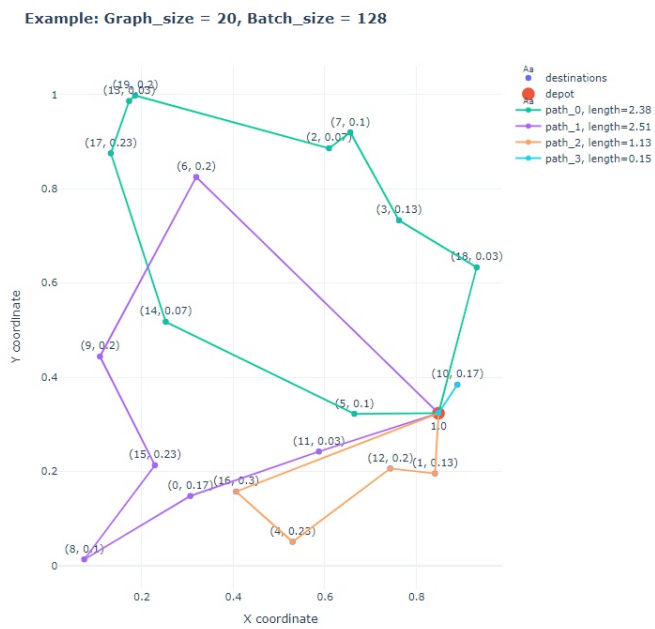
Example

Different models solving common instance:

```
Current cost:  6.04
Current path:  [0.0, 19.0, 4.0, 8.0, 3.0, 20.0, 14.0, 18.0, 11.0, 0.0, 7.0, 15.0, 10.0, 16.0, 9.0, 1.0, 0.0, 6.0, 12.0, 17.0,
5.0, 13.0, 2.0, 0.0]
```

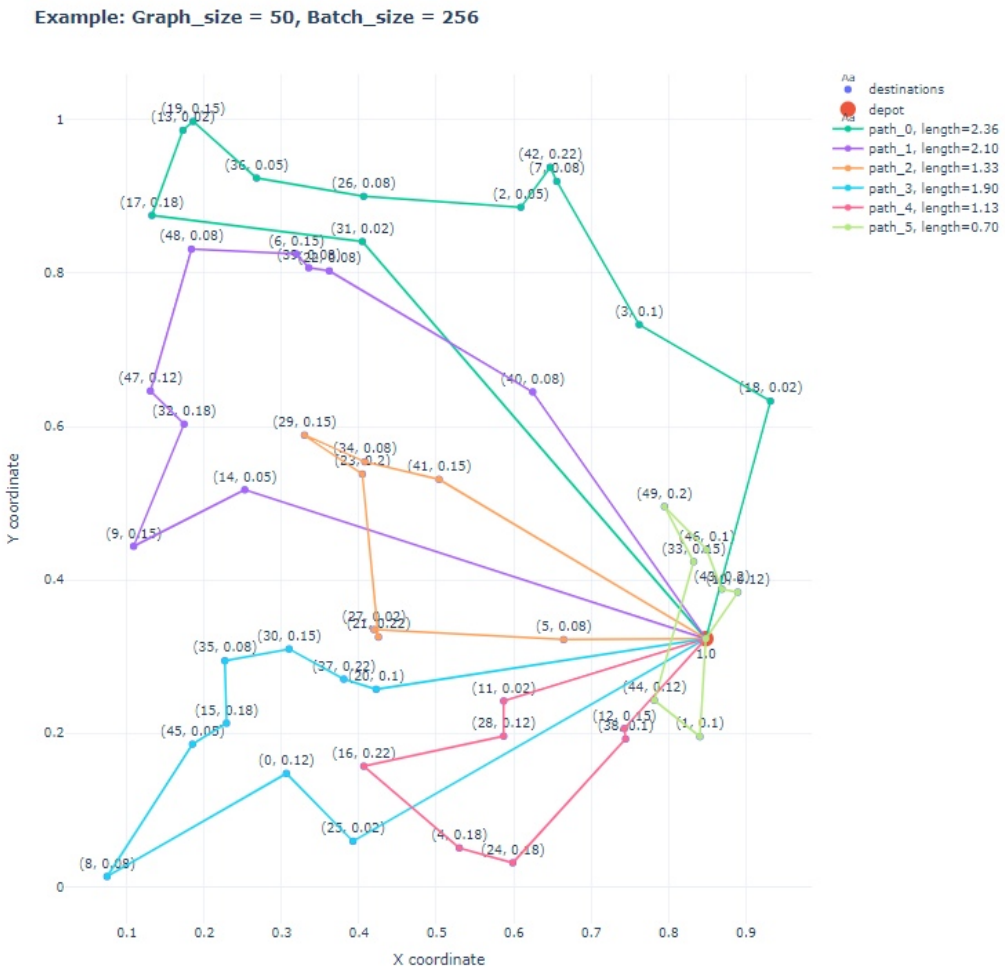



```
Current cost:  6.17
Current path:  [0.0, 19.0, 4.0, 8.0, 3.0, 20.0, 14.0, 18.0, 15.0, 6.0, 0.0, 7.0, 10.0, 16.0, 9.0, 1.0, 12.0, 0.0, 2.0, 13.0, 5.
0, 17.0, 0.0, 11.0, 0.0]
```



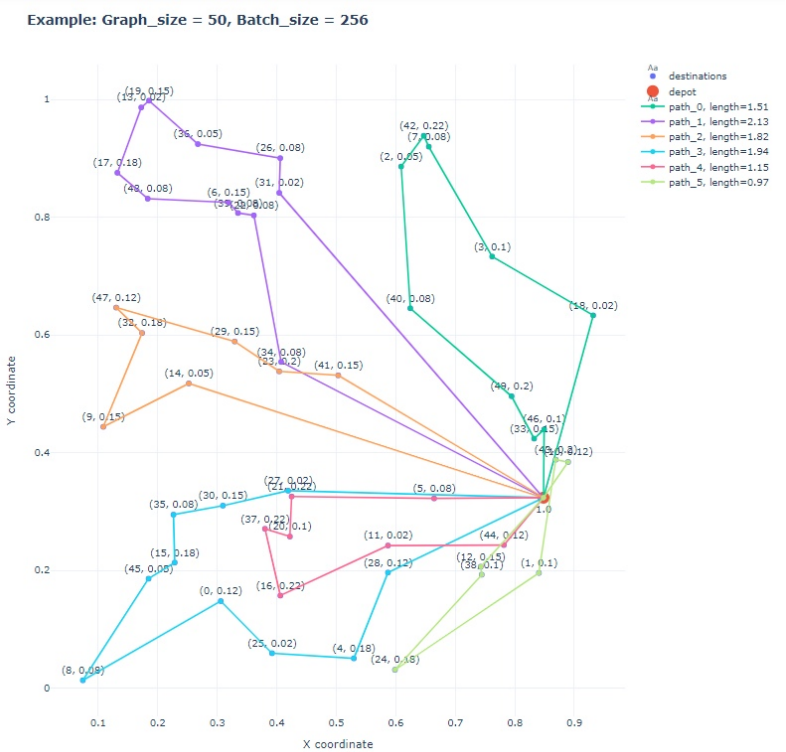
Example

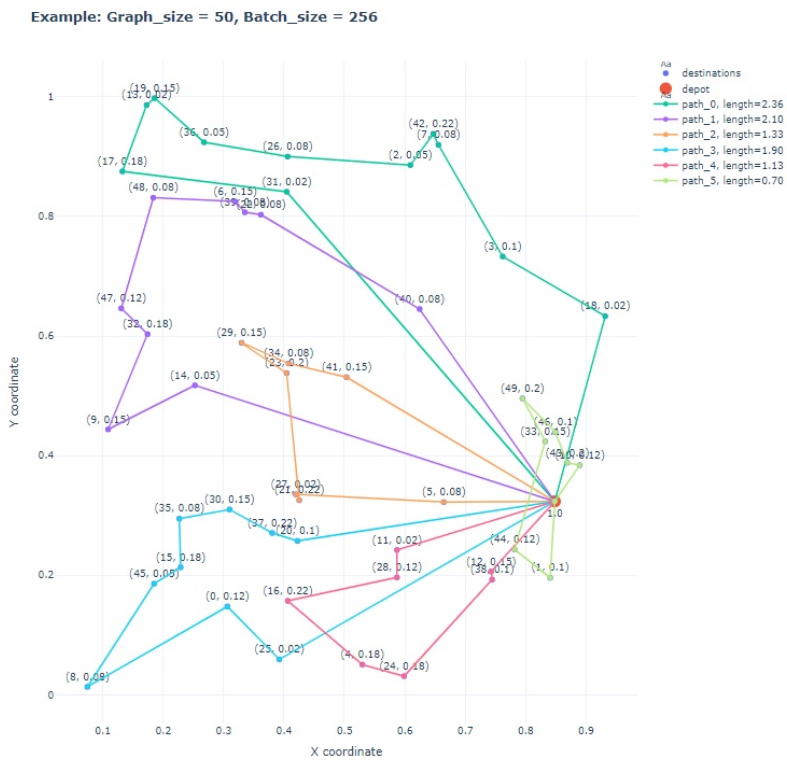
Solving VRP50



Example

Solving VRP50: notable changes from epoch = 33 to epoch = 53:





Thank you for your attention!