Capacitated Vehicle Routing Problem

Introduction

The Vehicle Routing Problem (VRP) in general terms is similar to Travelling Salesman Problem (TSP), which essentially has a graph where each node represents a location that must be visited. The nodes in the graph are connected via edge which essentially has the weight of distance between the nodes it connects. The goal of solving TSP is to minimise the total distance travelled, making sure we cover all the nodes. A modified and constrained version of TSP is the VRP. This has a graph similar to TSP, and additionally assumes a truck delivering goods to the nodes where each node has a specific demand. The goal of solving VRP is to minimise the total distance travelled, making sure that we deliver demands of all the nodes.

Though there are various versions of VRP, in this project we aim to provide a model to solve the constrained version of VRP, known as Capacitated Vehicle Routing Problem (CVRP). This has a limited truck capacity and will have to visit a depot to reload and make sure it supplies goods to other nodes.

Applications

The Capacitated Vehicle Routing problem is a prevalent business case in the field of Operations Research and Supply Chain Management. Finding the optimized path with respect to the constraints can help save companies' time and resources.

Related Work

Vehicle Routing Problem and Travelling Salesman Problem are considered as classic combinatorial optimisation problems in the sense that one must figure out the optimal ordering of nodes of the graph to be visited, with the goal of reducing the total distance travelled. There are many heuristic based approaches proposed to solve this combinatorial optimization problem. Recent research work and progress in the field of Reinforcement Learning proved to be promising to solve the combinatorial optimization problems. Various approaches which use Deep Q-Learning, Pointer Networks and Sequence to Sequence models were showcased in the recent publications.

In the article[1] by Julien Hernez, the author presents a unique and simple approach to solve TSP with Deep-Q Network. His work uses structure2vec to generate embeddings from graphs which

[1] "Routing Traveling Salesmen on Random Graphs Using ...." 15 Feb. 2020, https://medium.com/unit8-machine-learning-publication/routing-traveling-salesmen-on-random-graphs-using-reinforcement-learning-in-pytorch-7378e4814980.

form the basis of the Deep Q-Network. We use this as a basis for one of our approaches to solve the Capacitated Vehicle Routing Problem.

Implementation

We have implemented two different methods for experimentation.

1. CVRP with Graph Embedding using Deep Q-Network (DQN) Algorithm.
2. CVRP with CNN using Deep Q-Network (DQN) Algorithm.

In both the implementations we have observed differences in the result that we have obtained.

CVRP with Graph Embedding using DQN

In this approach, we implemented a Reinforcement Learning model with Deep Q-Learning for deciding future actions. To begin each episode of training, we start with a synthetically generated graph with a configurable number of customer nodes and depots. We also generate random demands for each of the customer nodes. We assume that the starting node is one among the depots and iteratively pick the next node to be visited until we satisfy the demands of all the nodes.

Similar to any Q-Learning approach, our goal is to train an action-value function Q(s,a) which gives the expected reward that can be obtained by performing action (a) when we are in state (s). In any particular state, we calculate the rewards of taking all possible actions and select the one which gives maximum reward. In our scenario, we define the state to be an object which has a list of nodes visited previously, demands of the nodes, total demand satisfied till now, capacity of truck left, locations of nodes, and pairwise distances of nodes. Since our next best move not only depends on the current node but also on the distances of all other nodes from the current node, we need an action-value function to consider the graph properties while calculating rewards. Hence, we need a network which takes graph details and generates an embedding to use it for training, apart from other features. Given that there are various graph embedding techniques, we also must make sure that our network is scalable to accommodate the varying number of nodes. Thus, we can avoid creating a new model for each configuration.

We have followed an implementation suggested in by Khalil Elias, et al. in Learning Combinatorial Optimization Algorithms over Graphs[2]. The authors in this paper use structure2vec[3], using which each node in the graph can be represented using a fixed size embedding. According to the research work of Khalil Elias, et al., an embedding of a node is generated by the following equation.

---

[2] "Learning Combinatorial Optimization Algorithms over Graphs." https://arxiv.org/abs/1704.01665
[3] "Discriminative Embeddings of Latent Variable Models for ...." 17 Mar. 2016, https://arxiv.org/abs/1603.05629

$$\mu_v^{(t+1)} \leftarrow relu(\theta_1 x_v + \theta_2 \sum_{u \in N(v)} \mu_u^{t} + \theta_3 \sum_{u \in N(v)} relu(\theta_4 w(v, u)))$$

The term $\mu_v^{(t+1)}$ is the embedding of node v generated after $t+1$ iterations. The term x_v is the vector of features of the node. In our case, this contains the information of whether the node has been visited, if the node is a depot, total demand left when the node has been visited, and the coordinates of the node. The second and third terms are neighbors' embeddings and a function of weights of edges at the node. Once we have the embeddings generated for a node, we then calculate the Q-values using the following learnable equation.

$$Q(s, v; \theta) = \theta_5^{T} relu([\theta_6 \sum_{u \in V} \mu_u^{T}, \theta_7 \mu_v^{T}])$$

Hence, the function $Q(s, v; \theta)$ is parameterized by theta values and it depends on the embedding of current node, as well as the aggregate of embeddings of all other nodes in the graph. Hence, we train the network to optimize the values of theta and maximise the reward.

In our experiment, we repeated the training process for 10000 episodes. At the start of each episode, we generate a random graph and corresponding demands and the initial node as one of the depots. During the initial episodes, we pick the next node randomly with a probability of ε and based on the reward values with a probability of (1-ε). As the number of episodes increases, we decay the epsilon and make sure that we select the next action based on Q-values and decrease the randomness. During the training process, we also maintain a history of events (states, reward and action) which are used to train the network. At each iteration, we sample a fixed number of past events and calculate the gradient of the error with respect to parameters of Q-function. We then update the parameters as we propagate this gradient through the network. We also checkpoint the model at regular intervals and denote by the median path length of recent path lengths. Once the training is completed, we can load the parameters of the checkpointed model and compute the results for test instances. We also make sure that the history of events is replaced with new incoming events which will reduce the probability of sampling the same data. One key consideration enforced in the model when we choose the next node is to update the truck capacity appropriately.

We consider the negative length of total distance travelled as a reward for a tour. We also enforce a mandatory reward if we visit a depot in between. This will make sure that the model learns to avoid depot while maximizing the number of nodes that can be covered with minimum travel distance.

Results

We have performed experiments by varying the number of customer nodes to 10, 20 and 50, which resulted in the following variation of total length of path as we progress through the episodes

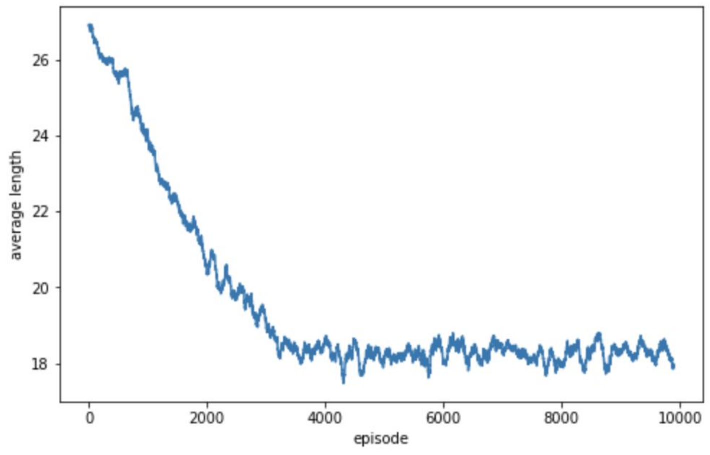

Figure 1. 50 Nodes

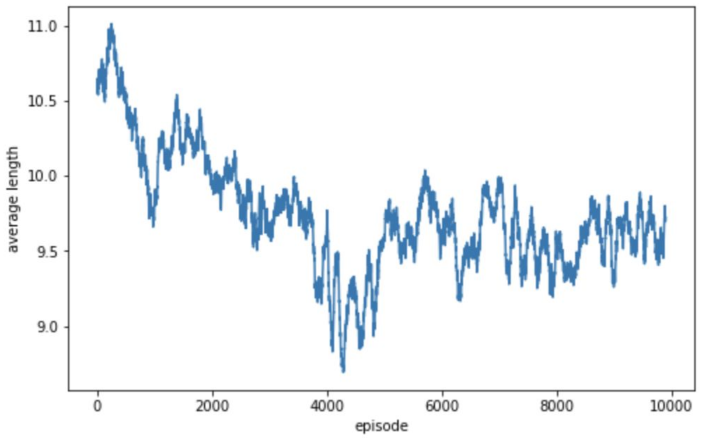

Figure 2. 20 Nodes

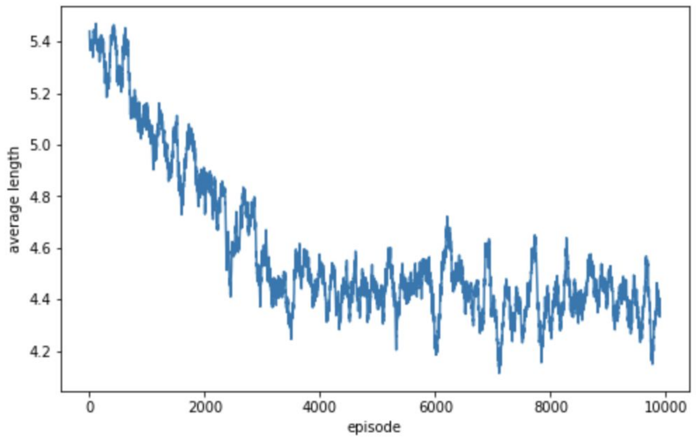

Figure 3. 10 Nodes

As expected, the path lengths during the initial episodes are higher as the model takes a random node as the next node along the path, and as the model is trained and the ε value decreases, we use the Q-Function to select the next best node to be visited, which results in lower path lengths where the model converges and the variance is constant.

Result Comparison with other Heuristics

10 nodes - Using config: truckCapacity = 100, numCust = 10, numDepot = 1, randomSeed = 1, numEpisode = 10000, embeddingDimension = 5, nnNodeDimension = 5, numQLStep = 2, gdBatchSize = 25, depotReward = 10

Sample	Path length - DQN Algorithm	Path Length - OR Tools	Path Length -Nearest insertion	Path Length -CW Savings
1	473.82	318	318	309
2	433.80	300	300	300
3	383.36	316	316	316
4	525.81	355	355	370

10 nodes - Using config: truckCapacity = 100, numCust = 10, numDepot = 1, randomSeed = 1, numEpisode = 10000, embeddingDimension = 3, nnNodeDimension = 5, numQLStep = 1, gdBatchSize = 25, depotReward = 10

Sample	Path length - DQN Algorithm	Path Length - OR Tools	Path Length -Nearest insertion	Path Length -CW Savings
1	428.41	309	309	309
2	445.94	285	285	285
3	310.76	223	223	223
4	330.52	249	249	249

20 nodes - Using config: truckCapacity = 100, numCust = 20, numDepot = 1, randomSeed = 1, numEpisode = 10000, embeddingDimension = 5, nnNodeDimension = 5, numQLStep = 2, gdBatchSize = 25, depotReward = 10

Sample	**Path length - DQN Algorithm**	**Path Length - OR Tools**	**Path Length -Nearest insertion**	**Path Length -CW Savings**
1	455.1	No solution	No solution	No solution
2	853.44	No solution	No solution	No solution
3	977.5	370	370	370
4	1077.43	410	410	424

*No solution because these algorithms do not consider going back to depot to refill capacity

50 nodes - Using config: truckCapacity = 250, numCust = 50, numDepot = 1, randomSeed = 1, numEpisode = 10000, embeddingDimension = 5, nnNodeDimension = 5, numQLStep = 2, gdBatchSize = 25, depotReward = 10

Sample	**Path length - DQN Algorithm**	**Path Length - OR Tools**	**Path Length -Nearest insertion**	**Path Length -CW Savings**
1	1919.5	No solution	No solution	No solution
2	2795.6	No solution	No solution	No solution
3	2488.7	628	626	626
4	2195.39	550	551	549

*No solution because these algorithms do not consider going back to depot to refill capacity

**Sample 3 and 4 capacity increased for OR Tools, nearest insertion and CW savings algorithm to obtain solution

CVRP with CNN using DQN

In this implementation we have used Deep Convolutional Neural Networks to implement the Deep Q-Network. The following is the algorithm for this implementation.
We have followed the process that is suggested by Jonathan Hui in his blog[4].
For the basic implementation we have referred to the blog written[5].

Algorithm

```
Read from the dataset
Create all truck related properties into Truck class //the delivery vehicle
Create all environment related properties into Environment Class //contains the rewards
Create all neural network related functions into NNModel Class
Initialize the truck object of Truck Class
Initialize the environment object of Environment Class
Initialize the deque replay_memory of size N
Initialize the main_model neural network
Initialize the train_model neural network
Initialize the epsilon_decay value
for episodes in (1, M) do
    Initialize episode_reward to 0
    Initialize step_number to 1
    Reset environment object
    Initialize done_flag to False
    while done_flag is True do          //taking the steps
        if random > epsilon then
            Select a random action
        else
            action is index of max of predicted values by main_model
        end if
        truck takes the action
        feed step related results into replay_memory
        Decay the epsilon value
    end while
end for
```

[4] "RL — DQN Deep Q-network - Jonathan Hui - Medium."
https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4.
[5] "Reinforcement Learning - Python Programming Tutorials."
https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/. Accessed 1 May. 2020.

Data set

The data set[6] used here are the files of format ***.vrp*** files which have the details of the capacitated vehicle routing problem. This problem can be extracted for the details and realised as a networkx[7] graph using the library ***tsplib95***[8].

In this implementation we have used the 0.6.1 version of the tsplib95 library, ignoring the latest update as the update was a major one, to version 0.7 in the middle of the implementation that introduced a lot of changes in the usage of the library.

The dataset has mainly scattered the nodes of the graph over a 100x100 matrix, and thus the coordinates of the nodes in a given problem are between 0 and 100. In addition to that, the datasets are for multiple delivery vehicles.

In the interest of time, the given dataset has been changed to have only one truck and the coordinates of the nodes have been made to fit a 10x10 matrix instead of 100x100.

Approaching the Problem

Visualization

In this implementation we are approaching the problem by realizing the given graph as an image to be fed into a Convolution Neural Network (CNN). The graph has not been realized directly as a networkx graph considering that we need the CNN to understand the variation visually in the state of the environment as the agent takes an action, so that appropriate rewards and penalties can be assigned. This can be implemented using ***opencv***[9] with much ease. Considering the given problem statement as a game would be helpful. So, consider the nodes that have zero demand to be in red, the nodes that have a demand to be green, the node where the truck is present as yellow, the nodes that have been already visited into white and finally, the depot being black. So, the game is essentially for the yellow node to only go to the green nodes and avoid red nodes with being able to go to the black node only when it is required.

For a ten node dataset that we have considered, the below is the visualization for the initial state, resized to 500x500. As we can see, here the third node is of demand zero.

[6] "Vehicle Routing Data Sets - COIN-OR." 3 Oct. 2003, https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm.old.

[7] "networkx · PyPI." 16 Oct. 2019, https://pypi.org/project/networkx/.

[8] "tsplib95 · PyPI." 18 Apr. 2020, https://pypi.org/project/tsplib95/.

[9] "opencv-python · PyPI." https://pypi.org/project/opencv-python/.

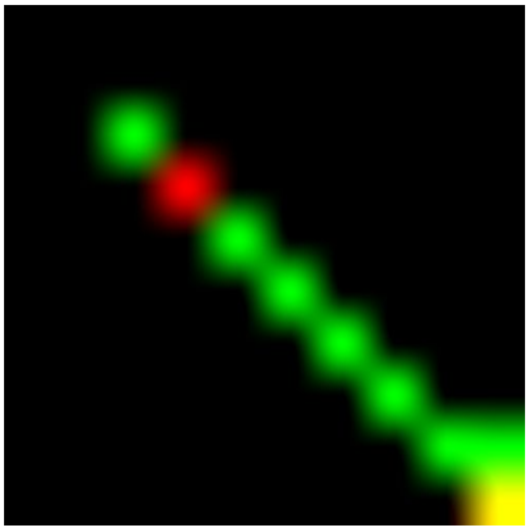

Figure 4. Visualization Step 1

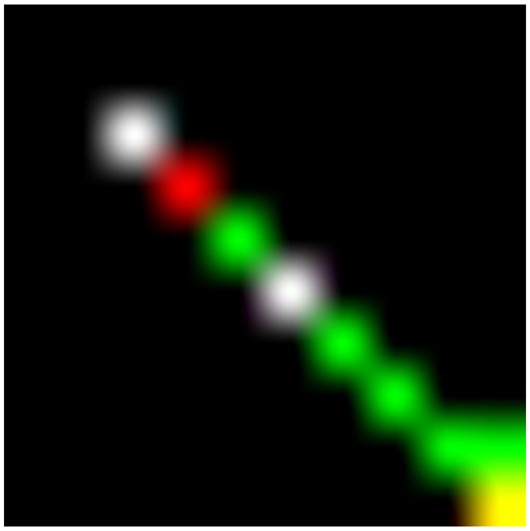

Figure 5. Visualization Step 2

After a couple of hops, the environment state is as it is to the right. As we can see, the nodes two and five have been already covered by the truck.
This image is fed to CNN for deep learning each step to make associations with the given rewards and penalties.

Epsilon

The epsilon value is being changed as a function of the episode number. The logic behind is that the system should have enough randomness in the beginning stages to be able to explore the system to find the optimal solutions, so that the later stages of the system can learn from them. The function is

$$\varepsilon = e^{(\text{-episode/epsilon\_decaying\_factor})}$$

Revisiting Depot to recharge Capacity

One of the aims of the project is for the system to learn when the vehicle needs to go to the depot to recharge its capacity to follow an optimum path. This is achieved by adding a penalty that is higher multiple times more than the other penalty to punish the system whenever the vehicle visits a node with a non-positive capacity.

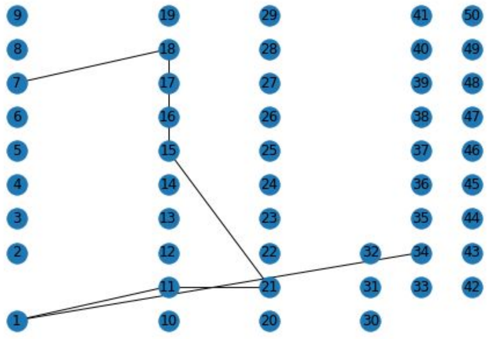

Figure 6. 50 Nodes - vehicle recharge

This is the path that is seen in the last episode of 50 nodes environment. The node 1 is the depot. As it can be seen from the image, the vehicle has gone to 1 to recharge.

Neural Network Architecture

Reinforcement Learning is like Supervised Learning in a way with the difference being its earlier outputs being fed into input again so that the system can learn. But even more important is the difference of having definitive and true training data for supervised learning which is missing in reinforcement learning. So, if we were to use only a model for the training and the prediction from the beginning, it would result in skewed results considering that in the beginning, the system is taking random actions, so, unstable.

So, to avoid this, we use two deep networks called *main* and *target* models. The main model is used to train the model and the target model is used to predict the action. After some defined number of steps, the target model's weights are updated to the main model's weights, so that the randomness that is inside the system does not impact the prediction.

We have used a Sequential Convolutional Neural Network for this implementation. There are three layers, with the first layer with 256 filters and the input to them with the size of the observation space of the environment, i.e. 10x10x3. The 10x10 is the 2D size of the image and the 3 is to accommodate the RGB color. There is also a maximum pooling size of (2,2) and a dropout of 0.2.

The second layer is the same as the first layer again with 256 filters and maximum pooling size of (2,2) and a dropout of 0.2.

The activation used is *Relu* and the optimizer used is *Adam*. The final layer of the network is of the output is of the size as the *action space* i.e. the number of nodes present, as the truck has the choice of selecting to go to any of the present nodes.

The output of the prediction consists of the Q-values of all of the actions that we have. We select the action that has the maximum Q-value.

Results

The aim of the project is to lower the total weight of the path that the vehicle takes to satisfy the demands of all the nodes.

Parameter Tuning

We consider the environment with 10 nodes.

Case 1

Here the epsilon decaying factor is 1000 with minimum epsilon 0.001, discount 0.9 and Adam optimizer learning rate is 0.001.

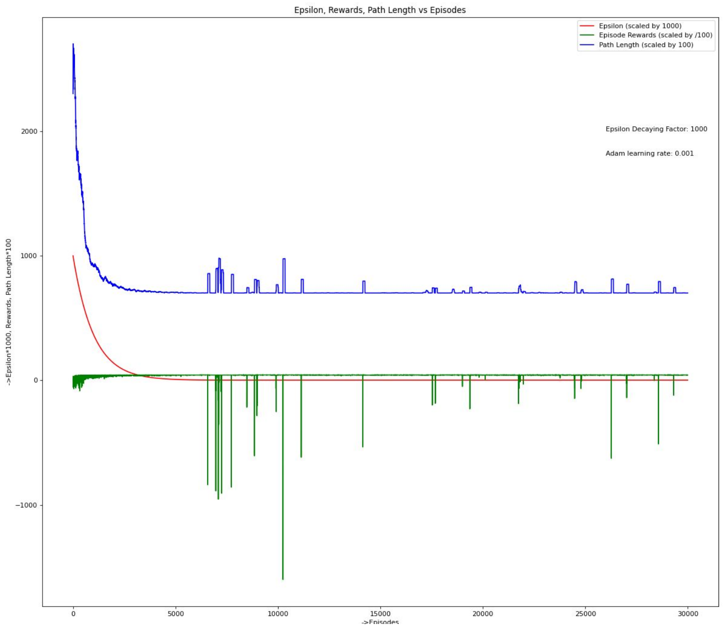

Figure 7. 10 Nodes Parameter Tuning Case 1

As it can be seen, the implementation is fairly normal with occasional anomalies where the system seems to be taking more than optimal steps, which is indicated by the sudden jump in the path length (blue) and the rewards (green).

The problem here is that there are still spikes that we would like to decrease in the system.

Case 2

Here the epsilon decaying factor is 1000 with minimum epsilon 0.001, discount 0.5 and Adam optimizer learning rate is 0.01.

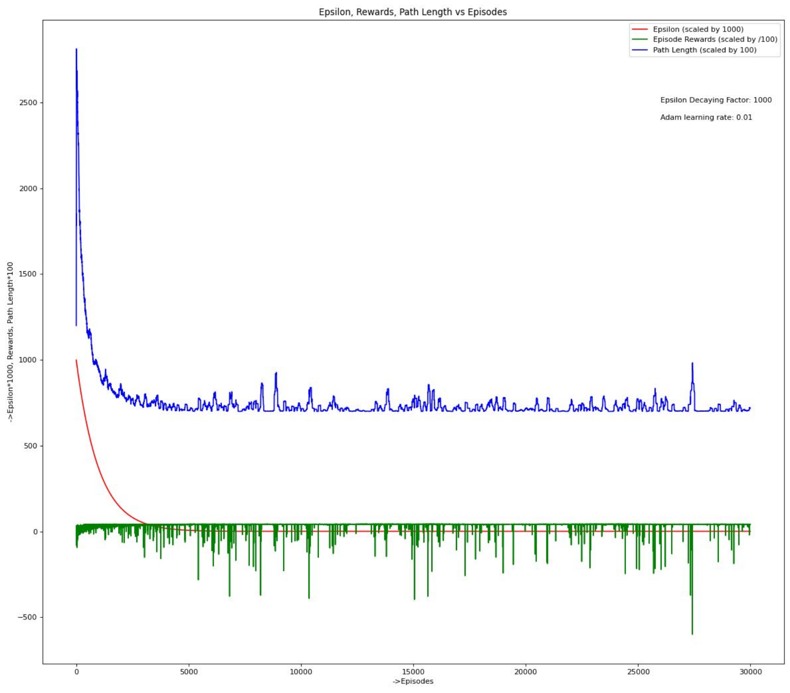

Figure 8. 10 Nodes Parameter Tuning Case 2

With the discount decreased to 0.5, the system does not care much about the future rewards as it does for the previous value of 0.9. So, as expected we can see more erratic behavior in the system with the random spikes that can be seen in the path length (blue) and the rewards (green).
It can be observed that the erratic behavior is still consistent throughout the system and that consistency comes because of the low epsilon value.

Case 3

Here the epsilon decaying factor is 1000 with minimum epsilon 0.1, discount 0.9 and Adam optimizer learning rate is 0.01.

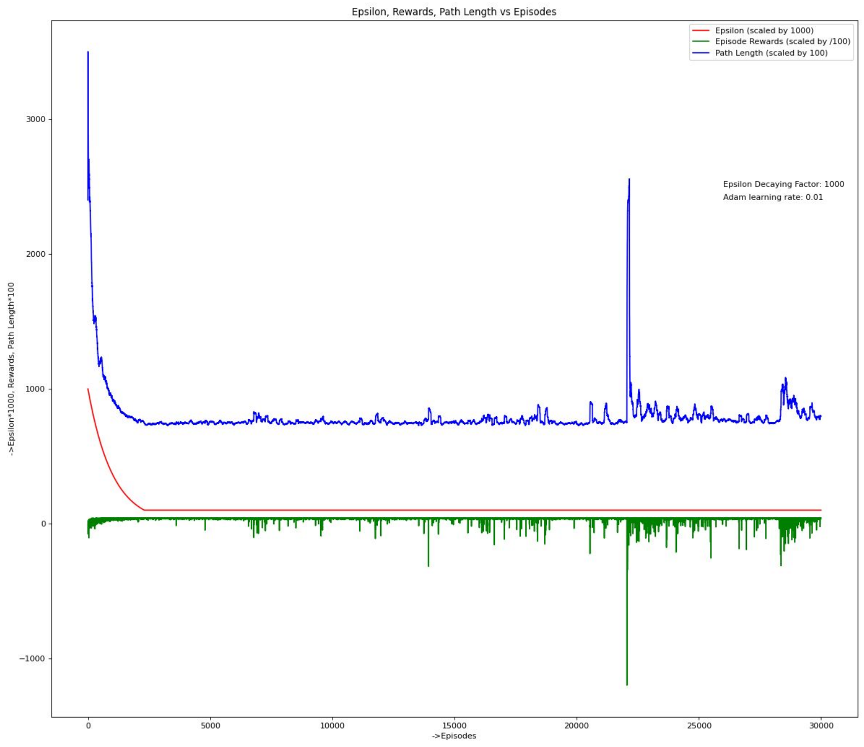

Figure 9. 10 Nodes Parameter Tuning Case 3

With the minimum possible epsilon increased to 0.1, the randomness in the system increases. So, as expected we can see sharp and sudden spikes in the graph with the path length (blue) and the rewards (green).

Individual Implementation

From the above parameter changes, we can conclude that the epsilon decay can be more than 1000 for better results, giving the system a few more episodes of randomness to learn from. In addition to that, we can also conclude that the learning rate of Adam optimizer can be 0.01, the minimum epsilon should be low, so 0.001 and discount should be high, so 0.9.

10 Nodes Implementation

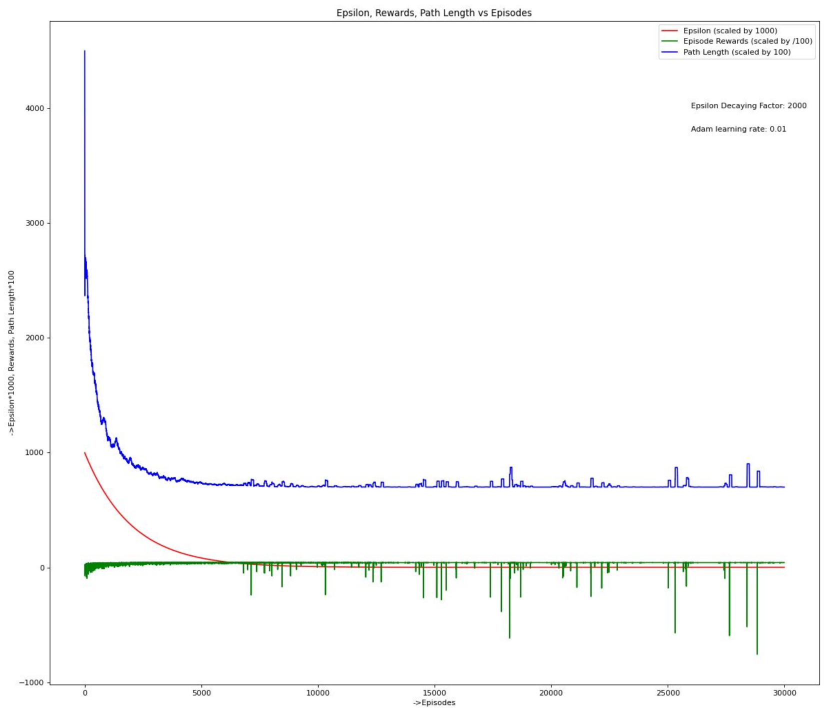

Figure 10. 10 Nodes

As it can be seen, the path length (in blue) decreases as the episodes increase.

32 Nodes Implementation

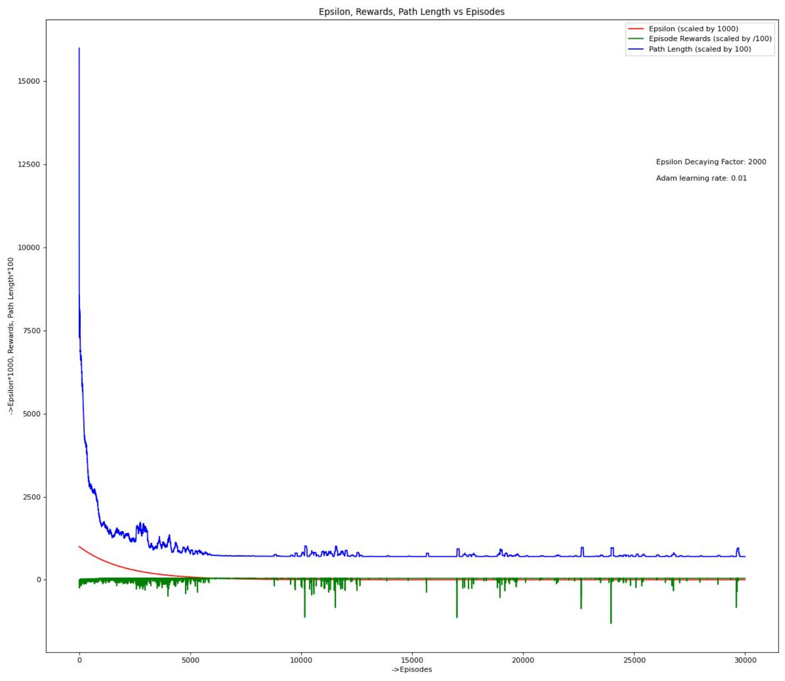

Figure 11. 32 Nodes

As it can be seen the path length is decreasing as the episodes increase.

50 Nodes Implementation

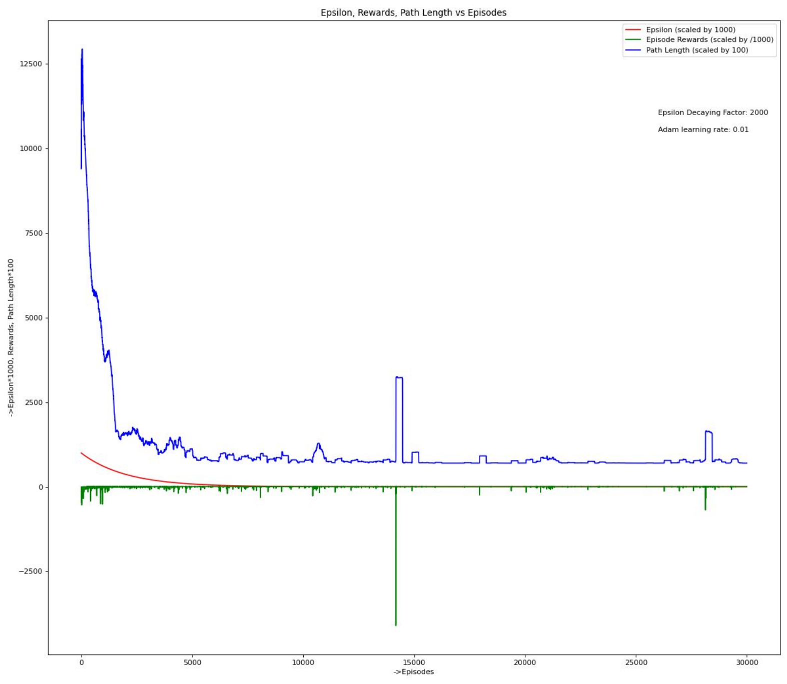

Figure 12. 50 Nodes

As it can be seen the path length (blue) is generally decreasing as the episodes increase, except the undesirable spike in the middle.

Conclusion

The vehicle routing problem is one of the most prevalent problems in supply chain management. Both the implementations to solve the Capacitated Vehicle Routing problem are not as optimized as Google - OR Tools or Clarke-Wright savings algorithm but comes pretty close. The one unique feature that has been implemented in both the implementations is returning back to the depot to replenish capacity if the sum of demands is greater than the capacity. Moving forward, the enhancements that can be done -- changing the features of each node in embedding to start with, using the information of truck capacity at the moment to train the Q-function and making rewards dynamic based on factors like number of nodes visited before visiting the depot.