ADBI CAPSTONE PROJECT
Travelling Salesman Problem with Deep Reinforcement Learning

<u>Introduction to the problem:</u>

Our problem is trying to solve the travelling salesman problem using deep reinforcement learning. We try to solve the problems where we need to visit every node in the graph with a constraint on fuel where we have a pre-set mileage on the truck that will need to refuel once the mileage distance runs out. We use the deep Q-learning approach with the struct2vec approach for graph embeddings to solve the problem.

<u>Related Work:</u>

TSP is a heavily researched optimization problem that is predominantly used for developing algorithms for fixed sized graphs that have trained knowledge about the structure and optimization of the problems.For example In Pointer Networks , Part of: Advances in Neural Information Processing Systems 28 (NIPS 2015) , where all neural nets were constructed using Pointer networks and they have a seperate layer for exploring

The relatively uncommon approach similar to the one used in this project is to use Reinforcement learning to let the algorithm understand about the problem in a way such that the knowledge of TSP is not inbuilt. Rather, it is learned using rewards.

Wide range of variants are also present for the Travelling Salesman Problem. Some examples include variation of metrics such as using other distance functions like triangular inequality rather than euclidean distance. And another variant is profit based travelling salesman problem.In the paper 'TRAVELING SALESMAN PROBLEMS WITH PROFITS: AN OVERVIEW' by Dominique Feillet et all., where the objective is to find a route with maximum profit and as per normal TSP minimising the distance travelled.The variation here is that all the cities need not be visited.

<u>Description of the datasets :</u>

Symmetric DataSet : [Train and Test]

For training the data we use a synthetic dataset with the help of the numpy random function. Using numpy random we build a Coordinate matrix of latitude and longitude values that represent the various nodes of the graph and based on this Coordinate Matrix we build a Distance Matrix that will hold the distance values between the various nodes of the Coordinate matrix.

We set the Episode values to 4000 which will train the model on 4000 different random generated graphs.

Real Data Set : [Test]

For testing, we have a test that can be run on a synthetic data set or a real dataset of cities. The real dataset has the following specifics: 1) 22 coordinates of Mediterranean cities 2) 52 coordinates of Berlin   3) 100 coordinates of real city(ref: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/)

<u>**Description of the specific problem:**</u>

Travelling Salesman Problem description:

The problem description involves solving the Travelling Salesman Problem of visiting every node in the graph in the shortest possible distance with a constraint of fuel. When the mileage restriction occurs there will be a search for the nearest fuel filling station which will be added to the tour to complete its path.

We are trying to solve the travelling salesman problem using the Deep Reinforcement learning approach that will be using the **Q-learning algorithm** to solve the problem.

We will be using the unsupervised version of learning for this approach.

Description of the specific algorithm [Q-learning with Struct2vec] :

We consider that the agent that we are building will have the full information of the environment and the states corresponding to its environment.
This is achieved by having a state in the graph that will hold the values of the coordinates matrix in place, the distance matrix that will be produced out of this coordinate matrix and the set of nodes that are already been visited. This makes the environment fully observable and trivial.
We will be using this state representation in the build of our Q-learning network.

Q-learning:

We try to develop a function Q(s,a) that will help choose the specific action for a particular state and return the rewards that would be obtained on doing this at the end of the episode.

Performing this for several episodes we would gather a good number of rewards and since it is a neural net we can differentiate based on the parameter of the function and use it to build a gradient of errors.

Using these gradient values we can improve our parameters of the entire function.
The basic approach of the function is to enable running of this function for several episodes and improve our parameters, on each run the algorithm will be able to choose its next best action using a greedy approach to help improve the functions parameters.

Occasionally based on the epsilon value the greedy approach would be replaced by a exploration of the graph randomly to help get out of a situation that will lead to a local optimal solution.

We also use the experience as a tuple to store values that help us in model learning. The experience will store the current state, the next action, the observer reward, and the next state that we can reach.

Struct2vec:
Used the struct2vec approach for the graph embedding process. In order to make our Q function be learning over different size of graphs and avoid the need of constructing a new model every time we choose a different graph size we have.
We have used the approach suggested in the paper : “Khalil, Elias, et al. “Learning combinatorial optimization algorithms over graphs.” NeurIPS. 2017.”
The approach tries to figure out a vector representation of each node.
We try to build a graph using the belief propagation that will send messages recursively to the other nodes along the graph that will help build the structure.

μ is the node embedding created for each node v, iterated over t times .
Term 1 : x represents the state of each node. Is a vector that has variables for capturing each variable defined in state
Term 2 : returns node embeddings for each of the neighbors(N) of v.
Term 3: edge weights of the incoming edges connecting to the current node v.

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

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

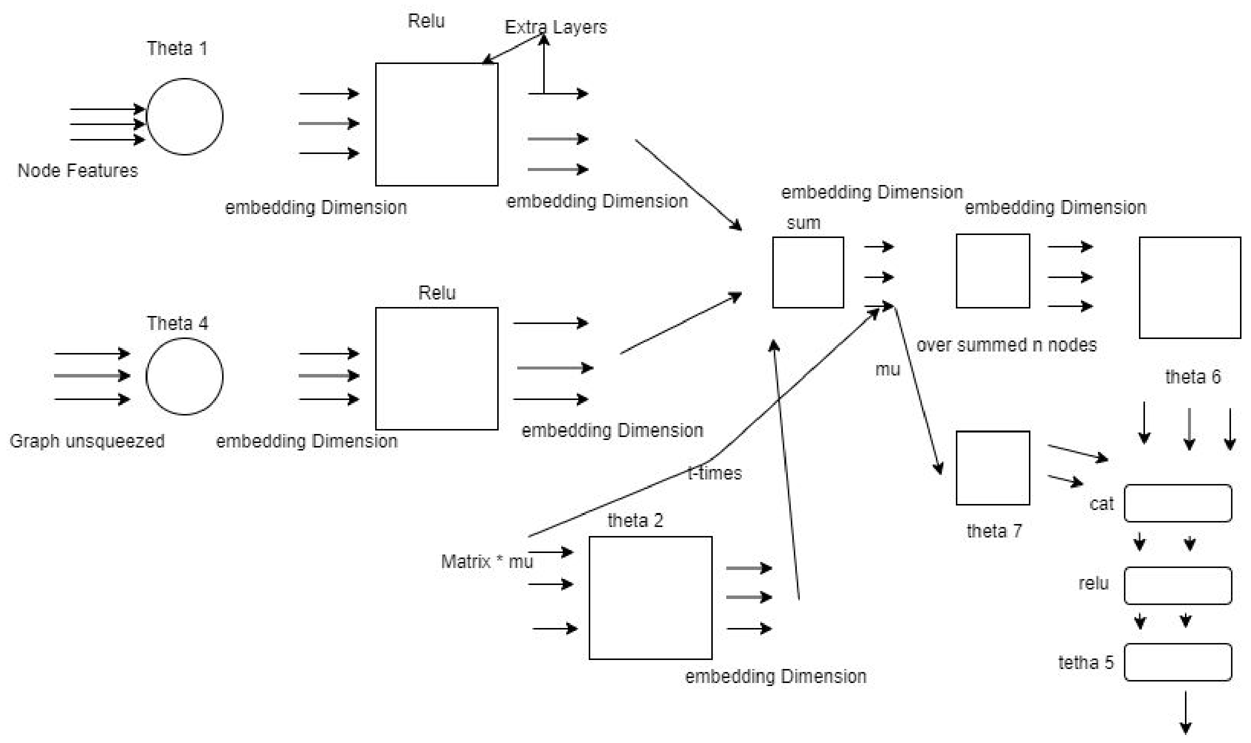
Q() function, for Θ ,which is the set of parameters θ
v which is the action represents the next node to be visited
Q() value is calculated by merging local embedding(u) and aggregation of embeddings of all nodes.

Architectures:

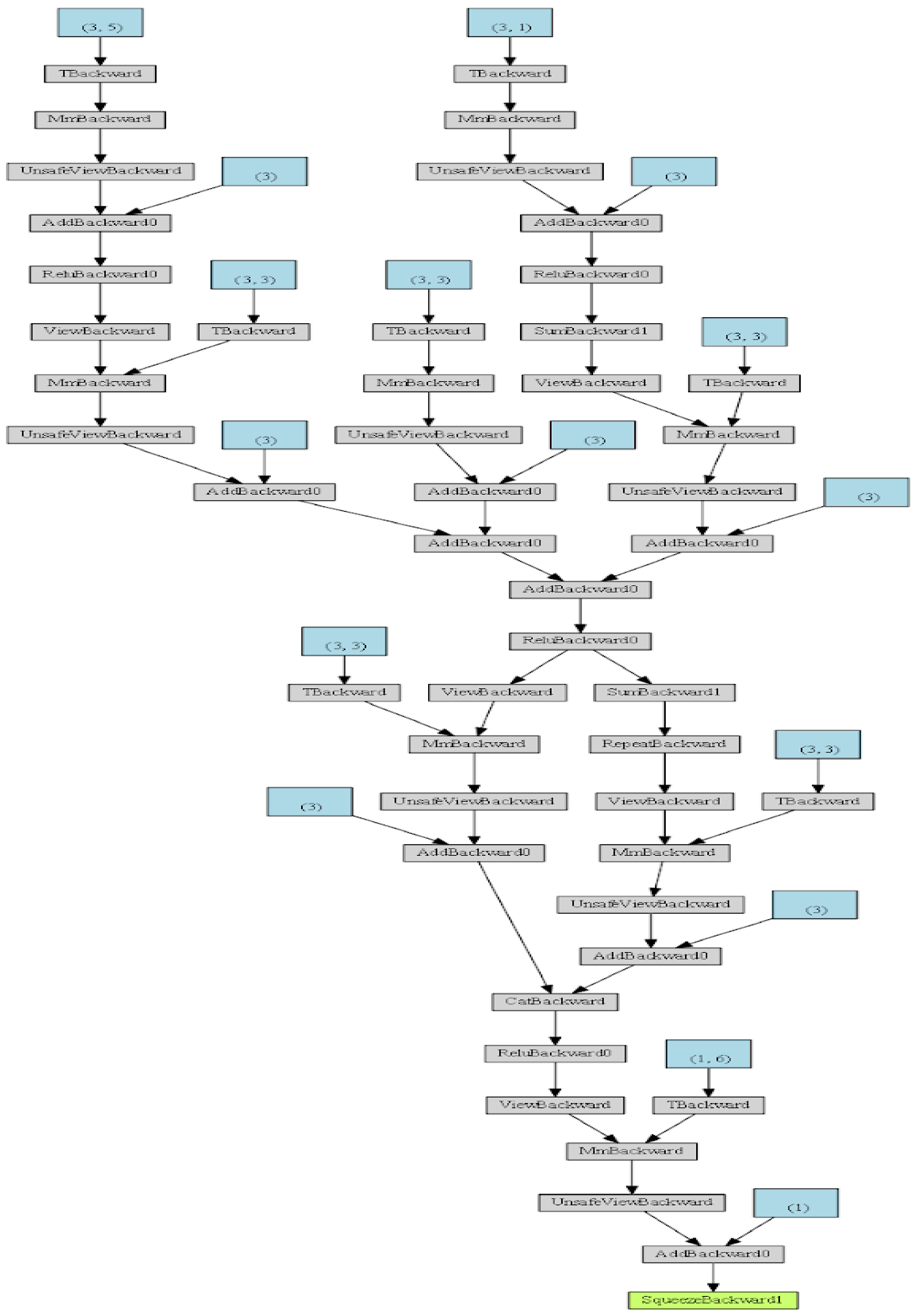
visual graphs :

Architecture of the Neural Net

The architecture of the neural network is shown below :



In the above diagram we see the various linear and relu layers that we have added to the system starting from the theta which takes node features as input and passes it to relu activation function with extra layers, parallely we can create theta 4 that will pass embedding dimensions to the relu function which is summed over with concatenation of matrices for matrix and mu and is summed over n nodes that is passed as theta 6 and and theta 7. Later we can concatenate them and use the relu activation function over it and pass it to theta 5 which will give our final output.



This is the entire flow of our deep learning model that is returned by the M_dot function.

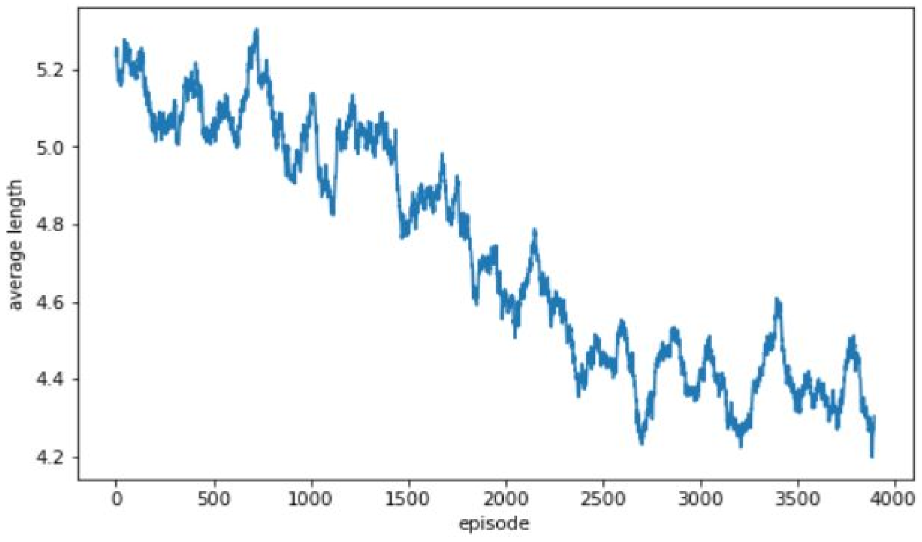
Experiments Results and Comparisons:

Architectures Hyper-parameters: table

The various hyperparameters that we have chosen are shown here :

SEED	1
GRAPH_NODE	10
EMBEDDIM	5
MEMORY	10000
QL_STEPS	2
GAMMAVAL	0.9
INIT_LR	5e-3
LR_DECAY_RATE	1. - 2e-5
MIN_EPSILON	0.1
EPSILON_DECAY_RATE	6e-4
NAME	'./models'
EPISODES	4000

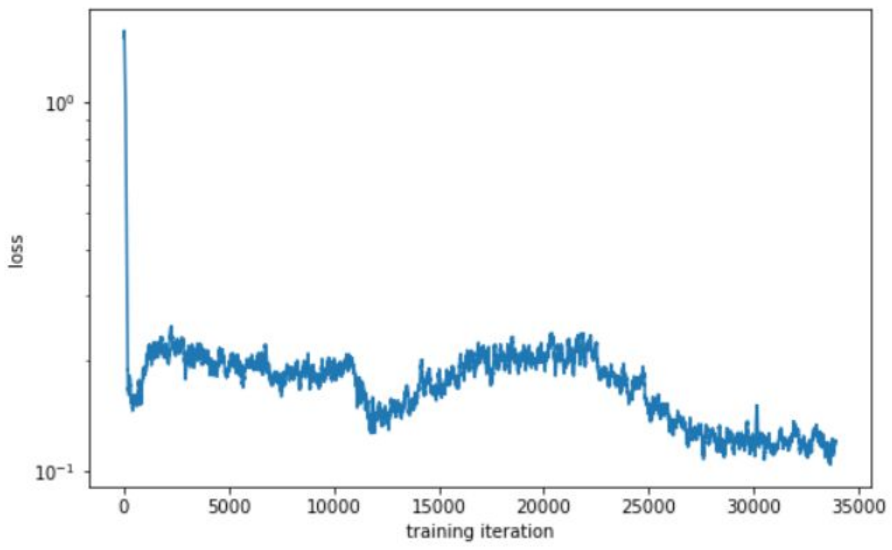
<u>Time/Epoch: graphs [Calculated as Moving Average]</u>



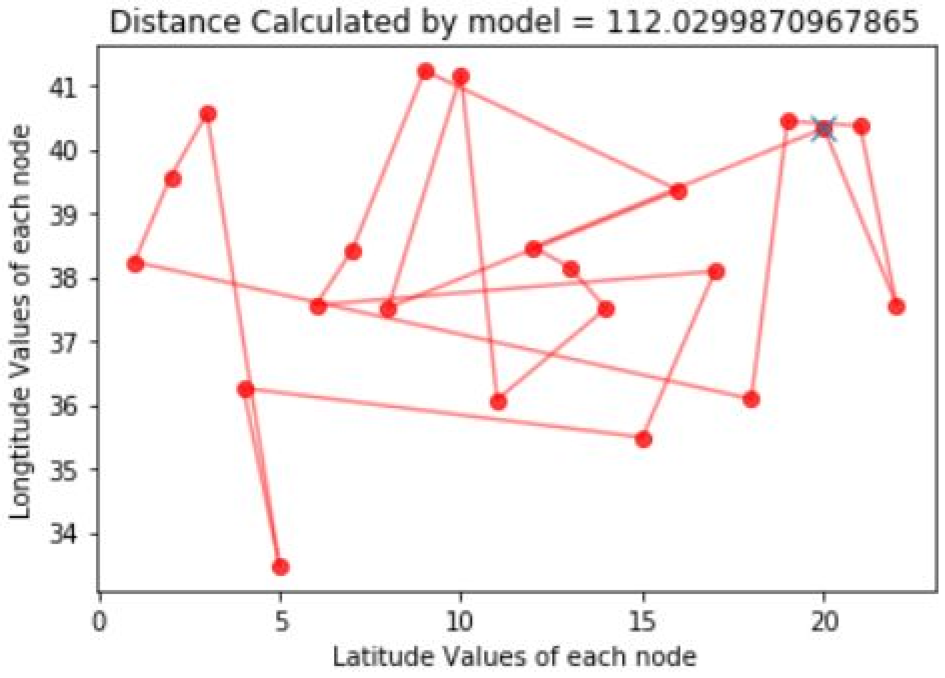
Here we can see that in the initial results with small episode values the average length being calculated is pretty high but as we move further we can see that this value gradually decreases and converges around the 2500- 3000 episode range where we get the average length calculated for the graph pretty low.

Loss Function Graph:

Here we can see how the initially high value of loss function gradually decreases.



Some of the Test values we recieved and the corresponding graph



Hyper-parameter Tuning:

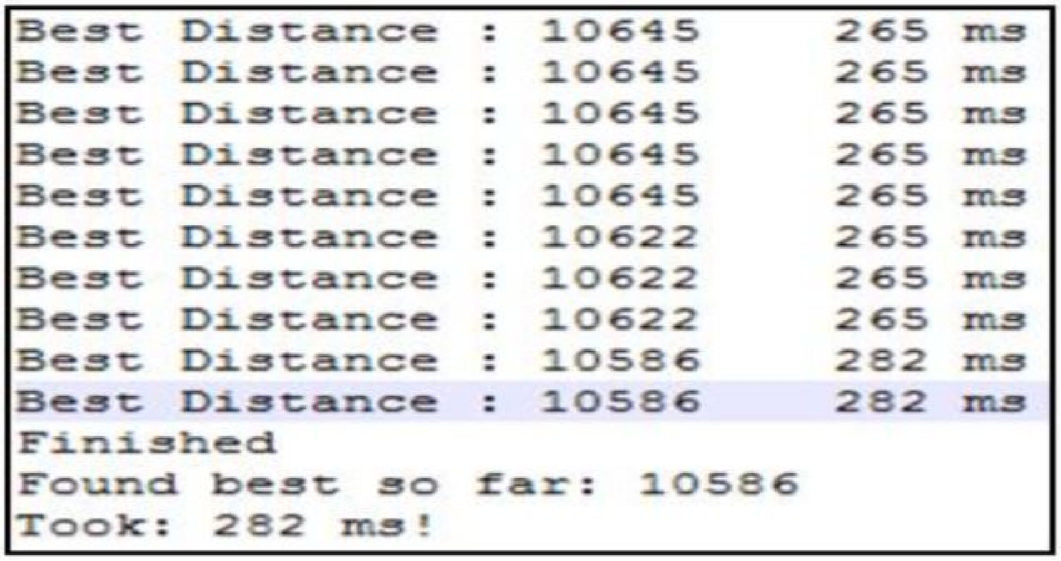
Choices, rationale :

SEED	set s the seed values for the random function
GRAPH_NODE	number of nodes in the graph
EMBEDDIM	dimension of graph embedding
MEMORY	memory function to store the experiences
QL_STEPS	number of steps in our Q-learning
GAMMAVAL	Discount factor
INIT_LR	The learning rate was set at 5e-3. This was chosen to make sure the number of episodes required was not being too large and also the convergence did not happen quickly
LR_DECAY_RATE	The decay rate was set at 1. - 2e-5, this was chosen to let the model take large initial steps and once the convergence starts to let it take smaller steps in the process. Impacted in reducing the loss in the function.
MIN_EPSILON	min value epsilon take
EPSILON_DECAY_RATE	epsilon decay rate
NAME	folder to store checkpoint models
EPISODES	iteration of the function run

<u>Comparison to other heuristics and algorithms and Results:</u>

The algorithm was tested with real dataset 'Berlin52' with 52 nodes and fixed coordinates of cities in Berlin city .ref:http://infonomics-society.org/wp-content/uploads/ijicr/published-papers/volume-6-2015/Performance-Comparison-of-Simulated-Annealing-GA-and-ACO-Applied-to-TSP.pdf)

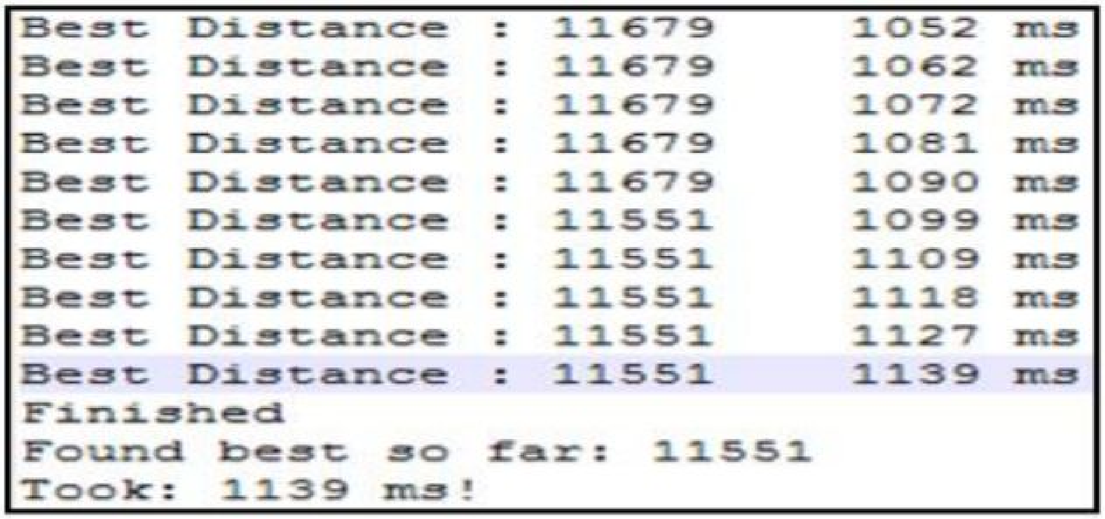
- **Simulated Annealing:**



```
Best Distance : 10645     265 ms
Best Distance : 10645     265 ms
Best Distance : 10645     265 ms
Best Distance : 10645     265 ms
Best Distance : 10645     265 ms
Best Distance : 10622     265 ms
Best Distance : 10622     265 ms
Best Distance : 10622     265 ms
Best Distance : 10586     282 ms
Best Distance : 10586     282 ms
Finished
Found best so far: 10586
Took: 282 ms!
```

Figure 10. Evolution of SA results over time in berlin52

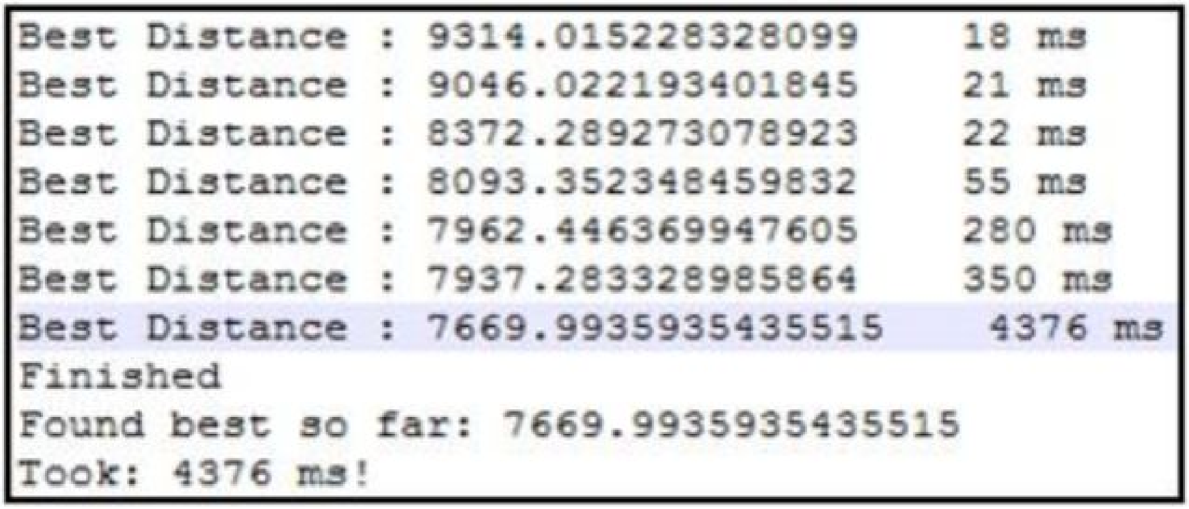
- **Genetic Algorithm:**



```
Best Distance : 11679      1052 ms
Best Distance : 11679      1062 ms
Best Distance : 11679      1072 ms
Best Distance : 11679      1081 ms
Best Distance : 11679      1090 ms
Best Distance : 11551      1099 ms
Best Distance : 11551      1109 ms
Best Distance : 11551      1118 ms
Best Distance : 11551      1127 ms
Best Distance : 11551      1139 ms
Finished
Found best so far: 11551
Took: 1139 ms!
```

Figure 7. Evolution of GA results over time in berlin52

- **Ant colony Optimization:**



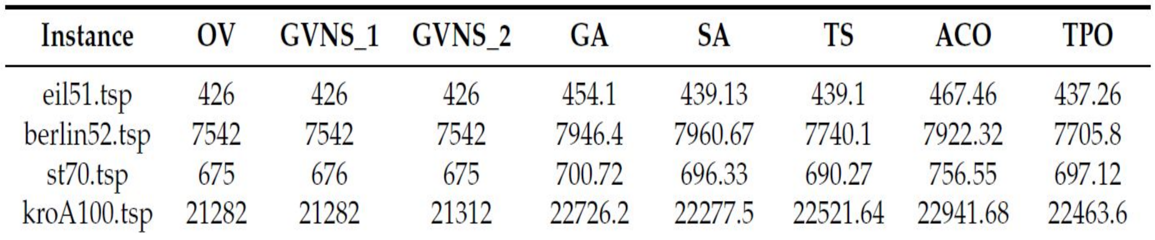
```
Best Distance : 9314.015228328099    18 ms
Best Distance : 9046.022193401845    21 ms
Best Distance : 8372.289273078923    22 ms
Best Distance : 8093.352348459832    55 ms
Best Distance : 7962.446369947605    280 ms
Best Distance : 7937.283328985864    350 ms
Best Distance : 7669.9935935435515    4376 ms
Finished
Found best so far: 7669.9935935435515
Took: 4376 ms!
```

Figure 4. Evolution of ACO results over time in berlin52

Comparison for berlin52 dataset:

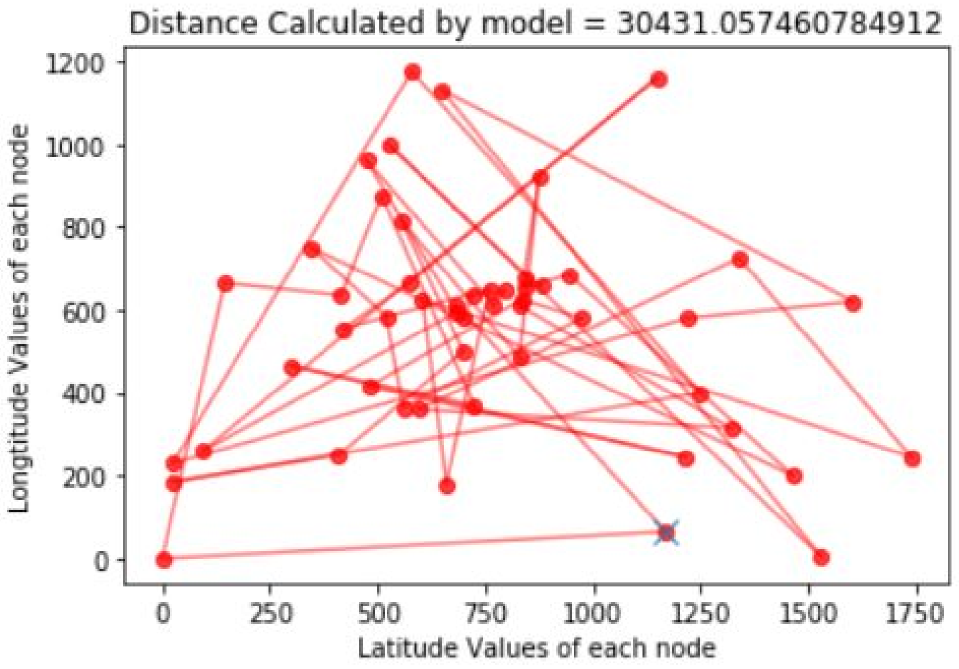
Simulated Annealing	Ant colony Optimization	Genetic Algorithm:	Q-learning with Struct2vec
10586	**7669**	**11551**	**15698**

The algorithm was tested for kro100 dataset with GVN algorithms for GVN_1,GVN_2 and GVN_3:ref:https://www.mdpi.com/2571-5577/2/4/31/pdf



Instance	OV	GVNS_1	GVNS_2	GA	SA	TS	ACO	TPO
eil51.tsp	426	426	426	454.1	439.13	439.1	467.46	437.26
berlin52.tsp	7542	7542	7542	7946.4	7960.67	7740.1	7922.32	7705.8
st70.tsp	675	676	675	700.72	696.33	690.27	756.55	697.12
kroA100.tsp	21282	21282	21312	22726.2	22277.5	22521.64	22941.68	22463.6

Q-learning with Struct2vec result:



GVNS_1 algorithm	**GVNS_2 algorithm**	**Genetic Algorithm:**	**Simulated Annealing**	**Tabu search**	**Tree Physiology Optimization**	**Q-learning with Struct2vec**
21282	**21312**	**22726.2**	**22277.5**	**22521.64**	**22941.68**	**30431**

<u>Conclusion:</u>

In this experiment we have used the combinatorial optimization of the Travelling Salesman Problem, using Q-learning and graph embedding using struct2vec approach. The choice for the algorithm is as described above.

The variant of TSP is adding a fuel station calculated as per the mileage hyper-parameter. And epsilon to take random solutions to explore the sample space.

The results obtained are given as graphs and in the comparative analysis as given above the results of the described algorithm is compared with other established approaches.We can see that from the results we are not able to match the current implementations in place but our algorithm is able to learning on graph of different sizes and implement a heuristic accordingly.

References :

1. Khalil, Elias, et al. “Learning combinatorial optimization algorithms over graphs.” NeurIPS. 2017.
2. https://medium.com/unit8-machine-learning-publication/routing-traveling-salesmen-on-random-graphs-using-reinforcement-learning-in-pytorch-7378e4814980
3. https://pytorch.org/docs/stable/cuda.html
4. http://people.duke.edu/~ccc14/sta-663-2016/03A_Numbers.html
5. http://people.duke.edu/~ccc14/sta-663-2016/03A_Numbers.html
6. https://stackoverflow.com/questions/15451958/simple-way-to-create-matrix-of-random-numbers
7. http://people.duke.edu/~ccc14/sta-663-2016/03A_Numbers.html
8. https://kite.com/python/answers/how-to-draw-a-line-between-two-points-in-matplotlib-in-python
9. https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.plot.html
10. https://therenegadecoder.com/code/how-to-check-if-a-list-is-empty-in-python
11. https://www.geeksforgeeks.org/namedtuple-in-python/
12. https://therenegadecoder.com/code/how-to-check-if-a-list-is-empty-in-python/
13. https://pytorch.org/docs/stable/nn.html

14. https://pytorch.org/docs/stable/nn.html
15. https://pytorch.org/docs/stable/torch.html
16. https://kite.com/python/docs/torch.ones_like
17. https://pytorch.org/cppdocs/api/function_namespacetorch_1a660c7003e58535ca935e0120c66a927e.html
18. https://discuss.pytorch.org/t/whats-the-difference-between-nn-relu-vs-f-relu/27599/3
19. https://stackoverflow.com/questions/44790670/torch-sum-a-tensor-along-an-axis
20. https://docs.scipy.org/doc/numpy/reference/generated/numpy.matmul.html
21. https://stackoverflow.com/questions/44790670/torch-sum-a-tensor-along-an-axis
22. https://jamesmccaffrey.wordpress.com/2019/07/02/the-pytorch-view-reshape-squeeze-and-flatten-functions/
23. https://stackoverflow.com/questions/53447345/pytorch-set-grad-enabledfalse-vs-with-no-grad
24. https://stackoverflow.com/questions/57237352/what-does-unsqueeze-do-in-pytorch
25. https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
26. https://stackoverflow.com/questions/52288635/how-to-use-torch-stack-function
27. https://www.geeksforgeeks.org/python-convert-a-list-into-a-tuple/
28. https://www.geeksforgeeks.org/python-random-sample-function/
29. https://numpy.org/doc/1.18/reference/generated/numpy.nonzero.html
30. https://www.geeksforgeeks.org/filter-in-python/
31. https://pytorch.org/docs/stable/optim.html
32. https://pytorch.org/docs/stable/optim.html
33. https://pytorch.org/tutorials/beginner/saving_loading_models.html
34. https://www.sharpsightlabs.com/blog/numpy-random-seed/
35. https://pytorch.org/docs/master/torch.html?highlight=manual_seed#torch.manual_seed
36. https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html
37. https://www.google.com/search?q=moving+average+plot+in+numpy&oq=moving+average+plot+in+numpy&aqs=chrome..69i57j33l2.8232j0j4&sourceid=chrome&ie=UTF-8#kpvalbx=_z56aXqW9KsW5tQaG95fQBA34
38. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.semilogy.html