Reinforcement Learning in IP Packet Routing

[1]Anik Pait (ID:1211225753), [2]Murtaza Rajagara (ID:1215188130), [3]Viken Shaumitra Parikh (ID: 1215126783)

Abstract: As the state space is growing exponentially day by day for plethora of activities, Reinforcement learning plays a crucial role in getting the right path with valuable states and solving the infinite state space problem. In this paper, we would see on how reinforcement learning behaves in the computer networks and handles the issue of packet routing among the routers. The Q Learning method derives the valuable state to pass the packet for the transmission to the destination node considering the Boltzmann Temperature as the factor to influence the actions committed by Q learning. On the latter case, the algorithm has been extended by optimization of Cooling function, searching for optimal learning rate and random exploration strategy.

Keywords: Reinforcement learning, Q learning, Packet Routing, Boltzmann Temperature

1. Introduction

Reinforcement learning transitions from the current state to the new state by a transition probability and earn 'rewards' for its actions and considers the same as an influence for the future actions. For a function Q, the Bellman equation for Q learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a} Q(s',a') - Q(s,a))$$

Where $s \rightarrow state$, $a \rightarrow action\ taken$, $\alpha \rightarrow learning\ rate$ and $\gamma \rightarrow future\ rewards$.

2. Literature Review

People over the past have been constantly working on improvements of Q learning function in some or the other way. Let's look at some of the earlier attempts made on the same. The first problem statement considered by R Rudek, L Koszalka and Pozniak Koszalka[1] was that the Q learning with the property task and representation takes $O(n^3)$ steps to reach the goal state. Also, node failures in a network often creates a deadlock in the transmission of the packets. Considering the deadlock, they proposed a Q learning model that considers a forgetting factor (Φ) to deal with the failure of the nodes without any additional channel to the routing network. Their proposed algorithm uses the standard Distance-Vector Routing (DVR), based on the Bellman-Ford Algorithm, which periodically updates routing tables using a cost factor. A node carrying the message/packet starts with an exploration strategy but as any tweaks in the network occurs, the algorithm adds up a forgetting factor Φ to the node without affecting the policy:

$$Q(s,a) \leftarrow \Phi.\, Q(s,a), \qquad \forall Q(s,a)$$

In the case of node failure, due to handshaking, the neighboring ports get cognizant about the same and the propagation doesn't take place through those nodes. The forgetting factor devaluates all the values of the routing table and lowers its importance to avoid the path. They have conducted various experiments to validate their algorithm including the fault detection task. Another problem statement which was considered by D Ganesha and Vijayakumar[2] is that the performance of a Q learning algorithm could be improved by improving the reward function. They proposed an algorithm to optimize the cumulative reward by creating constraint values K_i for every state to utilize the Q value. Each parameter K has its own 'M' restrictions such that:

S: K_1, $K_{2.....}K_n$ and $K_{i+1}\varepsilon(\max(K_i^{min}, K_i + step), \min(K_i^{max}, K_i + step))$

The learning with the training process doesn't operate on a optimal size compared to control guidelines. Hence, to learn the system better, they had introduced $P_{control}$ factor which includes the constraint values for every state for learning such that: $P_{learning}=\delta.P_{control,}$ $\delta\varepsilon N$, and the bonus operation Reward=90(1- Φ(t)) where Φ is the mean square error. Hence, after every reward it receives, it learns by a factor of $P_{learning}$ before updating the Q table for the next action it takes.

William Doan and Griffin Jarmin [3] considered a problem statement generally faced in the field

of robotics. The control algorithms in robotics includes the noisy and erroneous data due to its inability to handle non-linear environments and faces difficulty in forming the stochastic models. They implemented Simulation-based deep Q learning using transfer learning by getting a richer representation of the data. An experiment/simulation was conducted to train the agent with the CNN filters sizes changed according to the input size (image). The Q network was trained on image representation with approximately 900,000 iterations changing the weights at a certain point to observe the following results:

- Increase in average Q value with a drop in the cost function without impacting the system.
- The initial drop in average Q value due to the change of images but the training continued by the algorithm for the new images.

As reinforcement learning holds an important aspect in routing, researchers in Berkley [4] showed their concern on how routing configurations optimized with respect to traffic condition but couldn't achieve the expected results. They adopted deep reinforcement learning using Softmin routing to learn the per edge weights. The weight determines the shortest path between the two vertices. All the neighboring weights of the vertex was transformed into a probability distribution and could be interpreted as splitting ratios for the traffic destination. The reinforcement learning approach mapped the histories of the k traffic matrix to these weights to compute the max-link-utilization(congestion) of the resulting routing strategy for the next traffic matrix. The algorithm was compared to the non-ML algorithms and the performance of averaged congestion with the optimum congestion. It suggested that the proposed algorithm works better to make an efficient routing strategy for the next traffic matrix. Congestion is another major concern in routing algorithms and the problem was highlighted by Justin and Michael [5] in their paper. The Q learning learns the routing policies of different routing algorithms on minimizing the number of hops towards a congestion network by using local information only. But the approach is highly sensitive in a dynamic and irregular connection network pattern. They tested the algorithm across different network topologies and observed that the Q learning was able to provide efficient results at higher levels of network load rather than the shortest path algorithms. This paper suggested that if the Q learning is tailored specific to packet routing domain, it would provide better results. A similar observation was made by Sudhir K and Sharmila K [6] in their paper that RL based solutions provide better outcome in comparison to the static routing algorithms. Static algorithms fail to adapt the changing situations that brings traffic complexities while the RL based solutions adapt to these changes and minimize the average packet delivery time. They carried out a simulation where the node sends the packet to the nearest neighbor (less busy) and the neighbor updates the tables with the delays, rewards and penalties. The Hong Kong University researchers came up with a notion of predictive Q learning [7] to address the traffic congestion problem and inability to learn new policies by Q learning. The algorithm uses the Q values learnt to predict the traffic pattern. The congested paths need to have the packets sent at a significant frequency to avoid them for being candidates again. This exploration is controlled using a probing frequency that is estimated by the recovery rate of the path and yields better estimates for Q learning. As temperature forms an important parameter in routing, the factor was considered by Peter and Lazlo [8] that a relationship between simulated annealing and heuristic temperature bounds were shown. As exploration and exploitation tradeoff is always a dilemma, they showed that an efficient cooling function can be created by assigning the finite temperature values in the extreme cases. If temperature T is higher, the actions are uniformly distributed and if T approaches to zero, the highest Q valued action is selected. For K equally preferred actions, the probability of making those selections is K.

3. Benchmark Algorithm

Create a graph of nodes (routers) and its edges with the other nodes along with their weights

Initialize source and destination node (src_node,des_node) and the parameters such as learning rate, future reward rate

Set the Q table dictionary for all the node with an initial value except the destination (with 0).

```
def softmax(node, T):

    for i in q_tables[nodes].items():

            tp=(math.exp(i[1]/T))/math.ex
    p(i[1] / T)
```

Returns the best action based on softmax function (Boltzmann Temp)
def q_learning():
Calculates the q values and updates the old table for each iteration of the node
Check for convergence at each iteration
Select the best action among all the other actions
print the final q table for the nodes along with their q values

3.1 Q Routing in the Network Model:

The modelling of Actions and States: Actions are the 'delivering packets to the neighbor of the node' following a communication line. Rewards are negative values of delivery time. States are the source destination tuple. The number of nodes are 9 which were used in the baseline algorithm but in the further section of extension, we have implemented for higher number of nodes comprising in a larger topology.

SoftMax Action Selection using Boltzmann Temperature: Action selection is distribution dependent on the scalar temperature value T, which means agent choses an action according to the probability distribution defined by the Boltzmann's formula. Higher T values triggers exploration and lower T values exploitation happens. Routing table is equivalent to the Q table where routing table is maintained for each node and gets updated as it learns with the learning rate. The table contains IP address of the next hop and the probability of the action to be selected.

3.2 Results of the Benchmark Algorithm:

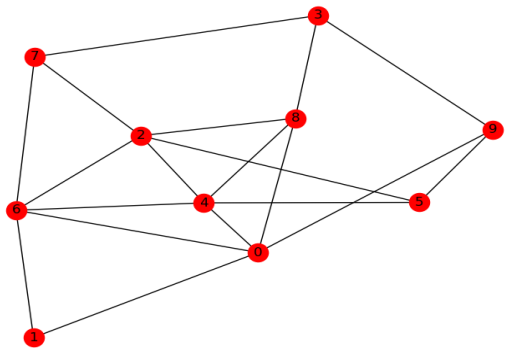

Fig 1: Graph of nodes with N (T=2000)

Let N be the number of vertices/nodes (routers) and be it as a connected graph with n edges. We had noticed that even for a similar and a larger topology, the behavior of Q routing remains the same as mentioned in the next section:

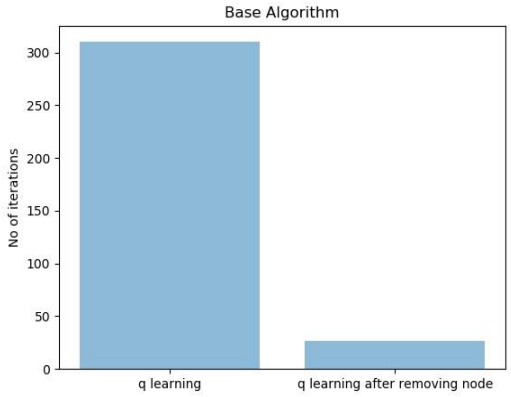

Fig 2: Iterations after removal of nodes

Performance of RL with no of nodes: As we noticed that as we remove the number of node links/ edges due to reason such as fault detection, the modified network uses the Q value to find another path and shows better results with a lesser number of iterations to converge. Hence, if a path break between any two nodes in the network occurs, the Q value gets updated on the table. The iterations are the average iterations took place after removing the nodes at different positions.

Assumptions: It is based on underlying assumption that each network output provides an action with the measure of agent's confidence. It is not a measure of optimality rather what agent feels can be an optimal action.

4. The Extensions to the algorithm

4.1 The Random 'Exploration' Strategy

In the random exploration strategy, we took the following considerations:

Approach:

- Instead of opting for the best Q value or the maximum Q value and going on with the greedy strategy, we tried a random strategy where we chose a random Q value as a part of exploration to go with it. All the information present in Q values are exploited.
- The actions are distributed over weighted probabilities. The random selection of action takes place irrespective of probabilities.

Assumptions: In circumstances where a random policy is optimal would this approach be ideal. However, it can be useful as an initial means of sampling from the state space in order to fill an experience buffer when using Deep Q Networks [9].

Results:

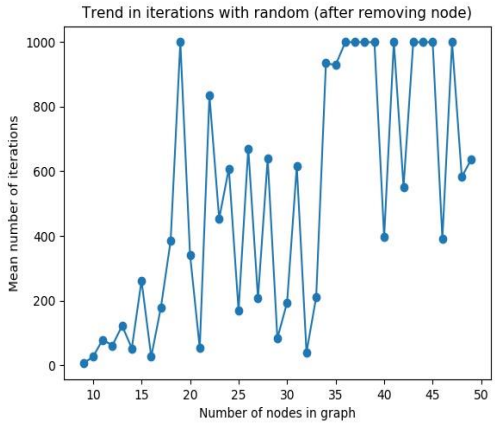

Fig 3: Random Strategy

As we removed the number of nodes, following a random strategy, where Q values were selected on a random basis and hence the iterations were fluctuating. Initially, the Q values converged at fewer iterations based on the lesser number of nodes but as the number of nodes increased, the path finding became a problem. The number of iterations were fluctuating in the topology. Hence, the random strategy exploration didn't give us a substantive response although for smaller topologies, random strategy can give up interesting and ideal exploration results. We stopped the iteration at 1000 and hence we can speculate how many iterations more it could have gone. Hence, the strategy can be useful in shorter networks but not an optimal strategy in larger topologies.

4.2 Gradually Optimal Learning Rate

Approach:

Different learning rates possessed different behavior. We tried different learning rates and observed when the number of iterations were optimal and found out that around 0.3, the learning rate becomes optimal with the number of iterations for even larger topologies.

Results:

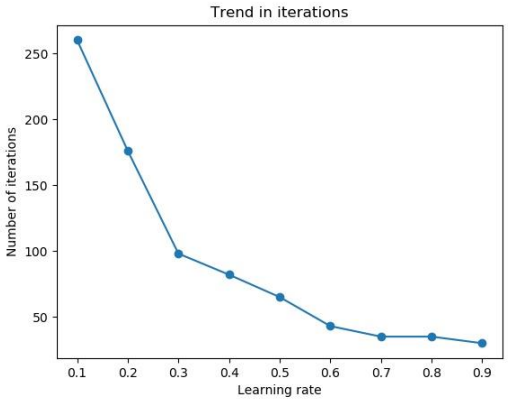

Fig 4: Learning rate

As we observe in Fig 4, As the learning rate increases, the iterations decrease in a uniform way

Learning rate = 0.1 (initially taken) and was gradually increased. If we see the initial value is resulting into higher

number of iterations to gain convergence.

In the next figure 5, as we started removing the nodes, the number of iterations decreased at much lower levels around 0.3 and convergence was successful at that rate compared to a full intact network. Although the number of iterations remain high at the initial stage but decreases gradually when learning rate is 0.2 and stabilizes at 0.3.

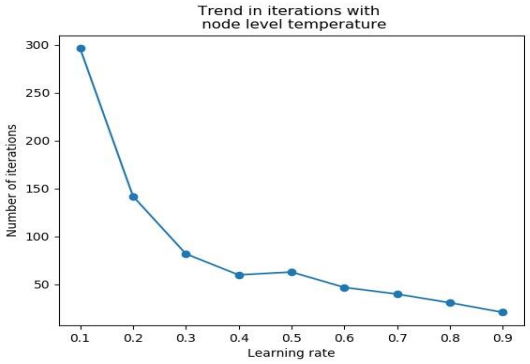

Fig 5: Learning rate after removal of nodes

4.3 Optimizing the Cooling function

The Boltzmann constant has been used to train our network decreasing it from the exploration to exploitation over number of iterations to decide the best path from source to destination.

Approach:

In the base paper approach, the Boltzmann constant (T) was used to vary over number of iterations from exploration to exploitation across all the nodes of the graph. In this strategy, emphasis is that in a graph large enough with greater number of nodes, we vary this temperature from exploration to exploitation across each node individually. This means that each node will have their own cooling function. This means that the nodes that have been traversed already on the graph will follow a greedy approach for selecting an optimal path compared to the other parts of the graph with nodes that have been traversed less in the previous trials and they will still follow the exploration strategy.

Results:

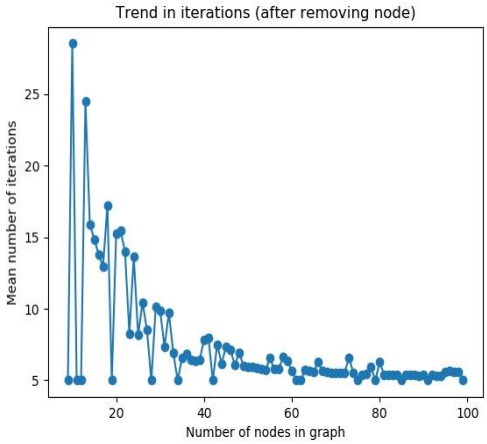

Fig 6: Trend in Number of iterations for retraining for the best path from source to destination

After a single node is removed on the graph v/s number of graph nodes increase varying same Boltzmann Constant across all nodes, figure 6 is observed. (maintain a global temperature of 2000)

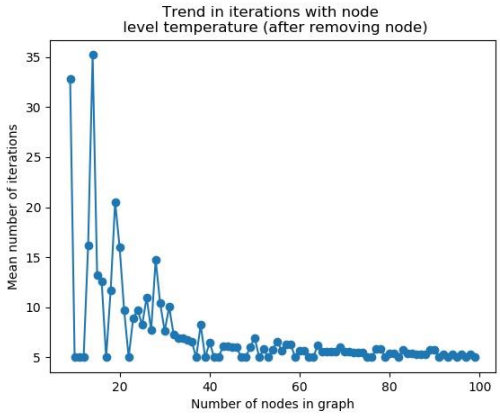

Fig 7: Trend in Number of iterations for retraining for the best path from source to destination

After a single node changes on the graph v/s number of graph nodes increase varying Boltzmann Constant for

individual nodes, the figure 7 is observed. (maintaining a local temperature across each of the nodes)

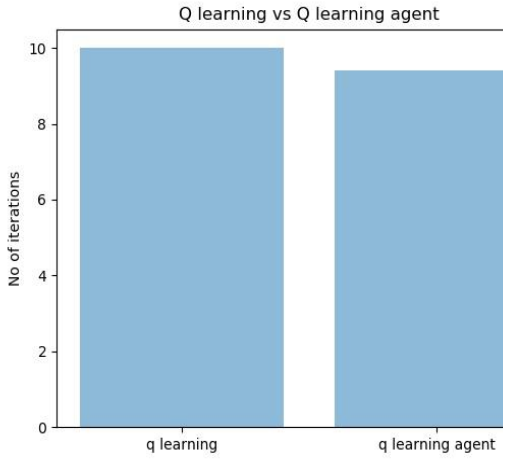

Fig 8: Comparison between average number of iterations for retraining for q learning in the proposed algorithm as compared to our proposed algorithm of varying Boltzmann constant across individual nodes

Assumptions: Varying the Boltzmann Temperature across each node independently doesn't affect the optimality of the solution.

We can see that the average number of iterations for retraining after node changes in Fig 8 are comparatively less in our proposed model as compared to the baseline implementation where we keep a global Boltzmann constant. One explanation for that is when each node that has been explored, the network tries to use the greedy approach in that part of the graph as compared to the other part of the graph where nodes haven't been explored and hence Boltzmann constant is still used to explore that part of the graph.

5. Conclusion

As we have noticed, how Boltzmann temperature affects the Q Routing strategy and reduces the number of iterations. The exploration and exploitation tradeoff are dependent on the Boltzmann temperature.
The cooling function and the learning rate places an important role in determining the Q Routing's behavior.
The future applications of the Reinforcement Routing can be applied to *Travelling Salesman Problem* and the *job scheduling problem* but with constraint that in such problems, all the nodes can be visited only once.

Code Execution Instructions

- The code has been written in python programming language. The code can be found in the code folder, the graphs that are generated are in the graphs folder and the plots for the various extensions are placed in the plots folder. The instruction for executing the code is to run the following command in terminal:
  " python filename.py "

- The name of the file with the corresponding function implemented in the file are:
  - *system.py* - The base paper implementation of qlearning with Boltzmann's temperature constant
  - *agent_t.py* - The implementation of Boltzman's temperature at node level
  - *random_algo.py* - The implementation of the random approach
  - *graph_generation.py* and *graph_misc.py* are additional and "optional" files to generate the graphs and plot the graphs. These are helper files to generate random graphs but for convenience we have saved the graphs_(number of nodes).json in the graphs folder to run on.

Acknowledgement

We would like to thank Professor Stephanie Gil for her constant guidance and support over our questions and our TA's Sushmita Bhattacharya and Weiying Wang for their prompt response to our queries.

References

(10 is the benchmark paper)

1. R Rudek, L Koszalka, Pozniak Koszalka, “Introduction to multi-agent modified Q learning routing for computer networks”, IEEE Xplore, 2005.
2. D. Ganesha, & Vijayakumar M.,” Implementation of modified Q learning technique in EMCAP control architecture”, Interational Journal of Engineering & Technology, Vol 7, No 1.5,2018.
3. William Doan & Griffin Jarman, “Simulated Transfer Learning Through Deep Reinforcement Learning”,2015.
4. Asaf Valadarsky, Michael Schapira,Dafna Shahaf, Aviv Tamar,”A Machine Learning Approach to Routing”, UC Berkeley, 2017.
5. Justin A Boyan & Michael L Littman,”Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach”, Carnegie Mellon University.
6. Sudhir K Routray & Sharmila K P.,”Routing in dynamically changing node location scenarios:A reinforcement learning approach”, IEEE Xplore,2017.
7. Samuel P.M., Choi & Dit Yan Yang, ”Predictive Q-Routing: A memory based Reinforcement Learning Approach to Adaptive Traffic Control”, Hong Kong University of Technology & Science, NIPS Proceedings,2017.
8. Peter Stefan & Laszlo Monostori, ” On the Relationship between Learning Capability and the Boltzmann formula”, Computer and Automation Research Institute,2002.
9. Arthur Juliani, “Simple Reinforcement Learning with Tensorflow Part 7: Action-Selection strategies for Exploration”, Nov 2016.
10. *Peter Stefan, Laszlo Monostori & Ferenc Erdelyi, “Reinforcement Learning for Solving Shortest-Path and Dynamic Scheduling Problems”, Computer and Automation Research Institute.