Centralized Traffic Assignment Control

FINAL PROJECT REPORT

Alireza Mostafizi

Introduction

The future of autonomous vehicles, routing behavior in particular, relies on centralized control to a certain extent. In this project I aimed to replicate the learning environment of the centralized control system to find the quickest path for the intelligent agents travelling along the transportation network using Q-learning algorithm. This project includes two major parts. In the first part, I focused on replicating the behavior of the controller in terms of learning the quickest path for a single intelligent vehicle in a simplified transportation grid network with the background traffic. Then I studied how different percentages of intelligent vehicles impacts the total mobility of the whole system, measured by summation and average of travel times. The results have shown that under the same travel demand, as the percentage of the intelligent agent increases in a specific network, the total travel time of the systems decreases, and increases afterwards.

The Problem

Imagine there is a network grid. Node represents intersections and link represents travel links. Nodes at the farthest left of the grid are origins and the other side nodes are representing destinations. In each origin there are 50 agents whose destinations are randomly assigned to one of the destinations. Two types of agents are included: naïve and intelligent. Naïve agents have no connection to the centralized control, and therefore choose their route based on the so-called diagonal shortest path as following.

$$Shortest\ Route = \begin{cases} \overbrace{R..R}^{\lfloor (6-D_y))/2 \rfloor} \quad \overbrace{RDRD..RDRD}^{d_y} \quad \overbrace{R..R}^{\lceil (6-D_y))/2 \rceil} & d_y < 0 \\ \underbrace{R..R}_{\lfloor (6-D_y))/2 \rfloor} \quad \underbrace{RURU..RURU}_{d_y} \quad \underbrace{R..R}_{\lceil (6-D_y))/2 \rceil} & d_y \geq 0 \end{cases}$$

Where $d_y = Y_{destination} - Y_{origin}$

R represents Right action

D represents Down action

U represents Up action

However, intelligent agents are connected to the controller that can provide them with the quickest path. Therefore, for each agent travelling along this network, the route would be combination of certain "Right" actions and $|d_y|$ "Up" or "Down" actions, based on the sign of d_y.

The questions are

- **PART I:** How a centralized controlling system can learn optimal quickest route?
- **PART II:** How does this machine achieves an optimal solution in the case that there are large number of intelligent agents trying to minimize their travel times?

Approach

The general approach to address this problem was to implement the Q-learning environment with reverse updates to speed up the goal based domain. The simulator was built in Netlogo and agent-based modeling was incorporated to replicate cars movement and speed synchronization as following:

$$Speed = \begin{cases} Speed\ of\ front\ vehicle - deceleration & If\ there\ is\ any\ vehicle\ in\ sight \\ Speed + acceleration & If\ the\ is\ no\ vehicle\ in\ sight \end{cases}$$

Where $acceleration$ and $deceleration$ is set to be 0.01 and 0.05 respectively. The dimensions are Patch per tick which are the distance and time dimensions in Netlogo.

Since traffic congestions are simulated in the model and incorporated in the process of learning, rewards were stochastic, each trajectory was run 10 times and the average rewards of each state were used to update the Q matrix.

The approach is summarized in the following steps.

1. Start with all zeros initial Q Matrix
2. Take the action considering $\epsilon - greedy$ exploration exploitation policy
3. Collect the reward
4. Go back to step 2 until reaching the destination
5. Do the same trajectory 10 times
6. Update Q function in a reverse order based on average observed rewards in 10 trials of the very trajectory as following
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \beta\ Max_{\acute{a}}\ Q(\acute{s},\acute{a}) - Q(s,a))$$
7. Go to 2: After trying 10 different trajectories, calculate and report the performance measurement.

In addition, to formulate this problem, three key features of reinforcement learning are defined as following:

- **States**

  Each intersection is known as a state.
- **Actions**

  For the first Part, possible actions are D and R, regarding the network constraints. This means that an agent cannot step outside the network.

  For the second part, possible actions are R, and either D or U (based on the sign of d_y), Considering the constraint that an agent cannot step outside the network. Also, an agent cannot pass its destination vertically. This means that any sets of actions will take any agent to its own destination for sure.
- **Rewards**

Since the objective is to find the "Quickest" path, reward of each state is based on the time to get to that state from previous state formulated as following.

$$Reward(state) = (\ Max_{Traveltime} - Traveltime(state)\ ) * 1000$$

Where $Max_{Traveltime}$ is calculated based on the link length and the minimum speed of the vehicles. In the simulation it is set to be 470 since I was using Netlogo ticks as time measurement. There also is a huge reward for the destination, based on the time it takes to get to destination from origin as following

$$Final\ Reward = (\ Max_{Total\ Traveltime} - Total\ Traveltime\ ) * 1000$$

Where $Max_{Total\ Traveltime}$ is calculated based on Manhattan distance between origin and destination and the minimum speed of the agents.

Part I

For all purposes of this project, I considered two different network sizes, three by three and six by six, to reduce the amount of computation needs, and prove the concept.

In this case all agents are naïve, except one intelligent agent at the top left of the grid, heading to bottom right. Based on travel pattern proposed above, we should be expecting background traffic more or less as following:

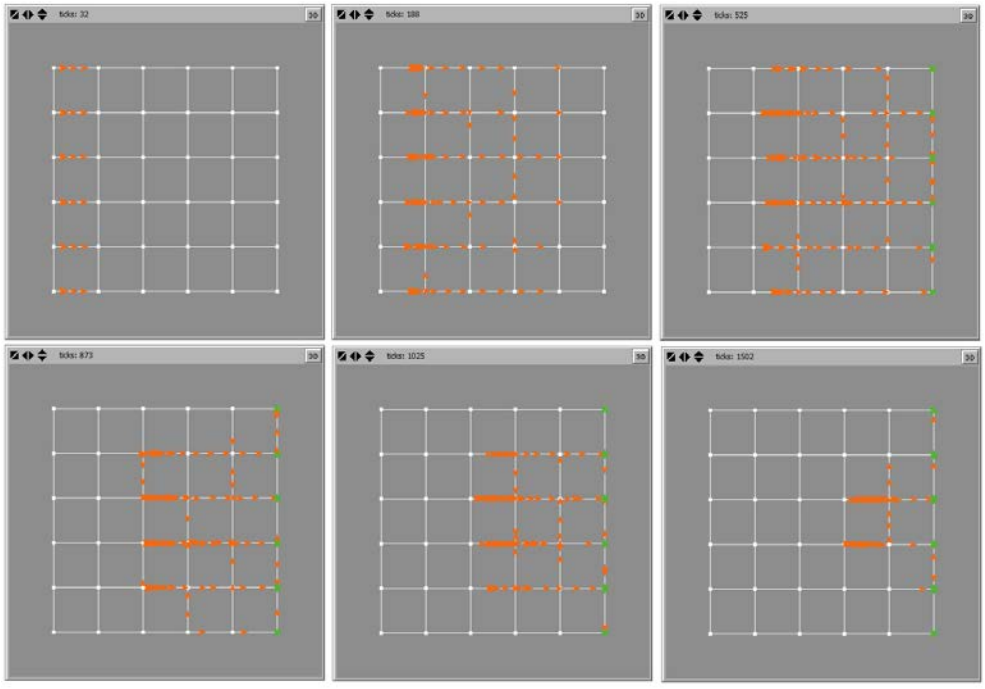


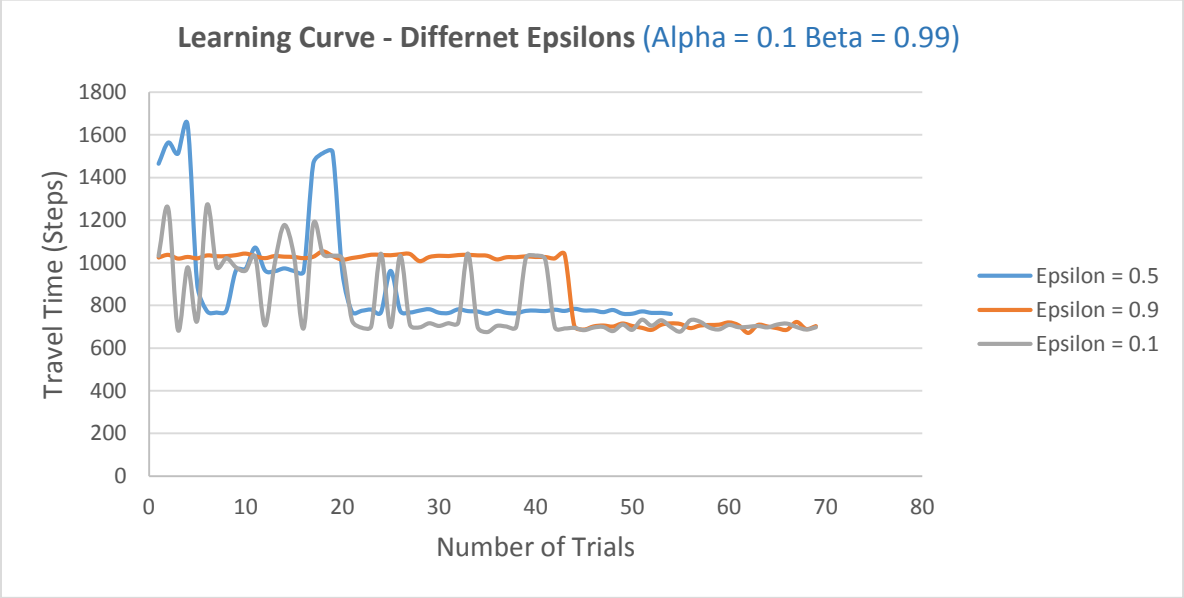
Figure 1. Background Traffic Patterns

As shown in the figure above, the middle of the network is pretty congested and quickest path should avoid those links as much as possible. Thus the quickest path from the top left to the bottom right is not the diagonal shortest path, and it is more likely to be inclined to the bottom left of the grid. Also, since vertical links are less congested in the beginning, it is expected the quickest path to be more vertical in the beginning and after that, it tries to get to the destination by moving horizontally.

To compare the impact of different coefficients, I set the learning environment for two different network sizes, and for deferent values of β, α, and ϵ for the epsilon greedy exploration exploitation policy.

Results

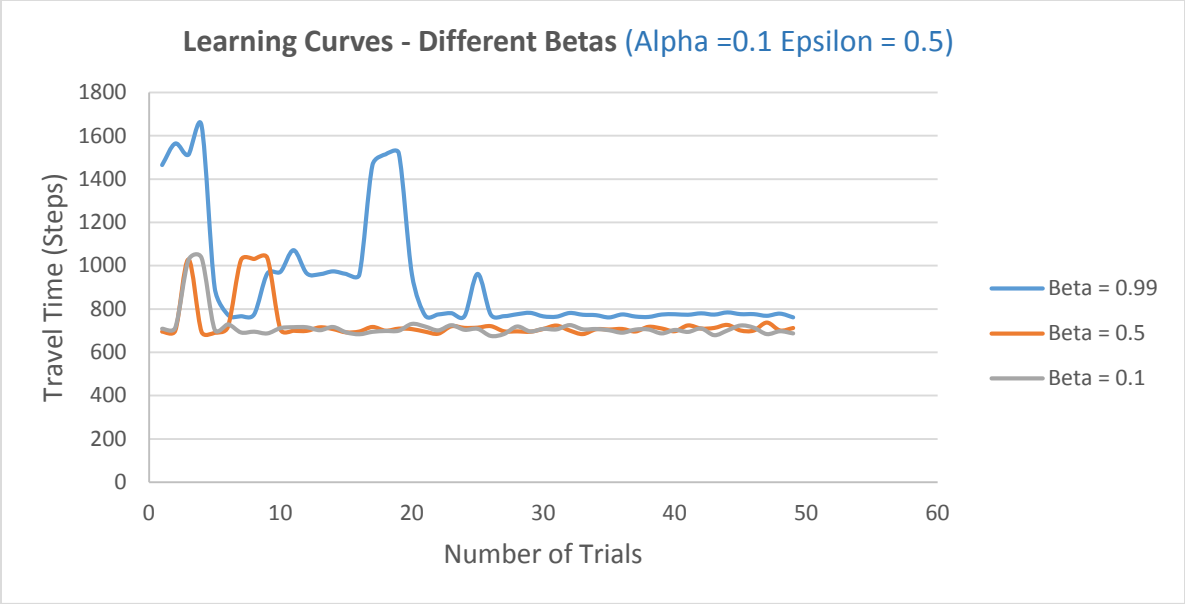
The following charts shows different performances of the algorithm due to different coefficients on a three by three network. In the first graph, I tested the algorithm with different values for ϵ.



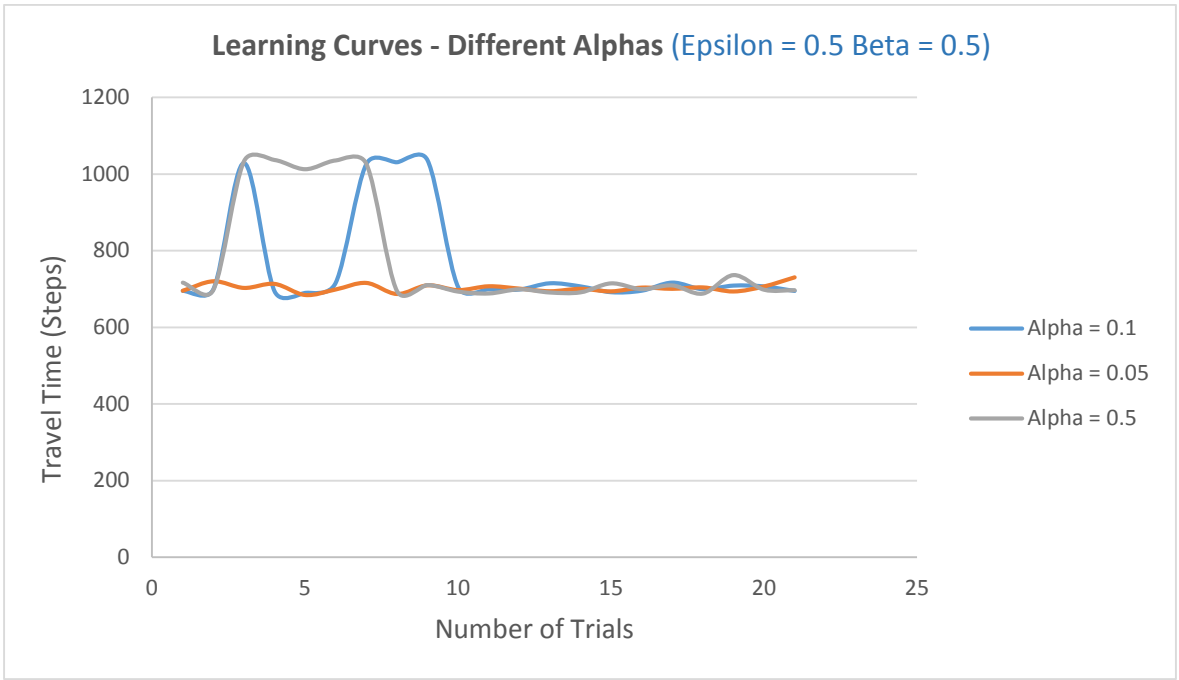
Considering the background traffic, the optimal route would be the one that goes the bottom left intersection, and due to the nature of the domain, it is hard to explore the corners of the network. So, setting ϵ to 0.9, even though it makes the learning curve less noisy, it would converge to the near-optimal route eventually but in a very long time. On the other end, setting it to 0.1, makes the learning curve very noisy, and reliable convergence to the optimal route happens almost at the same time as 0.9-greedy policy converges. Therefore, for the sake of this domain, and since I only cared about convergence to the optimal policy, not the way of convergence, I set ϵ to 0.5 which practically is random exploration policy.

The next graph represents how different discount factors affect the performance of the algorithm. Since the rewards of each state are in line with the final reward, having a large β sometimes makes the learning curves to be noisy. As you can see in the nest graph, all three curves show that the algorithm converged to the near-optimal policy. However, $\beta = 0.99$ in a noisier way.

For the sake of this project, to make sure that the algorithm convergence to the quickest route, I set β to 0.99.



Next graph demonstrates the impact of different values of α on the algorithm performance. As expected, the less the alpha is, the smoother the learning process would be.



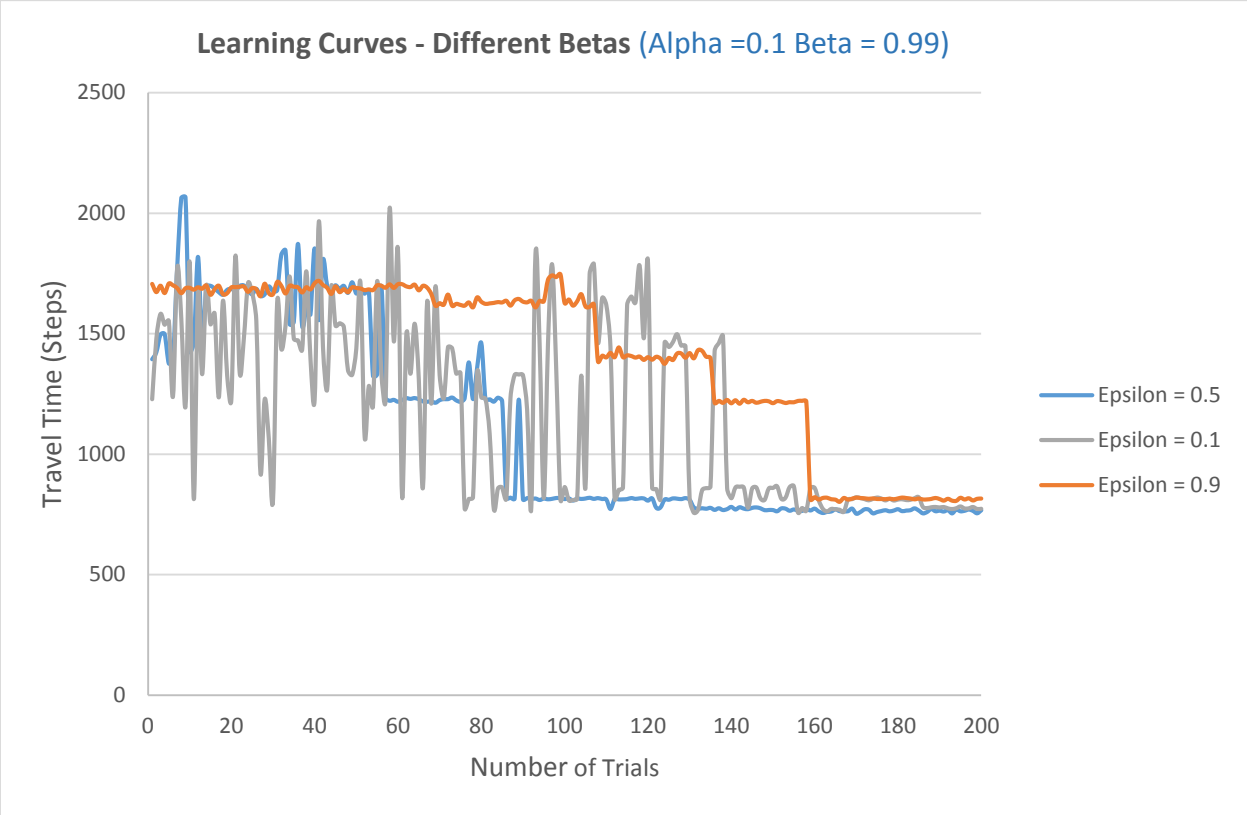
Therefore, for analysis in the second part of the project I chose these coefficients to be as following:

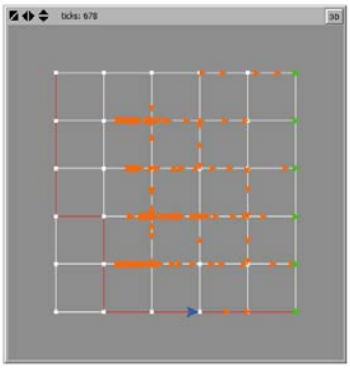
$$\alpha = 0.1$$

$$\beta = 0.99$$

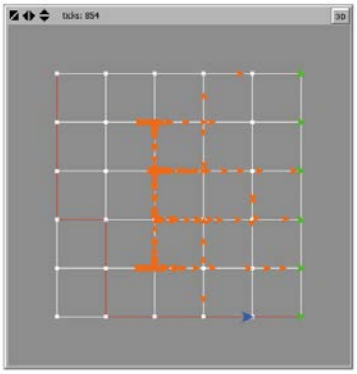
$$\epsilon = 0.5$$

The final result in the first part is the sample of algorithm run for a six by six network. As observed, similar to the smaller network, ϵ of 0.5 converges faster, but noisier, than the other two. You also can capture the final route came from the algorithms in figure 2.

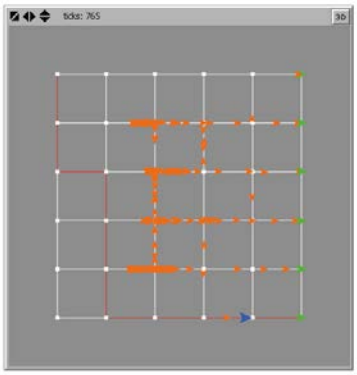




0.1-Greedy



0.5-Greedy



0.9-Greedy

Figure 2. Final optimal routes

Part II

In this part, I focused on having multiple agents learn at the same time. As before, there are 50 agents in each origin. However, this time a specific percentage of the agents in each origin is set as intelligent. The destinations are assigned randomly. Definition of States and rewards are exactly the same as the first part. However, possible actions are defined as following.

If $d_y > 0$

$$Posssible\ Actions(state) = \begin{cases} R \\ U \end{cases}$$

If $d_y < 0$

$$Posssible\ Actions(state) = \begin{cases} R \\ D \end{cases}$$

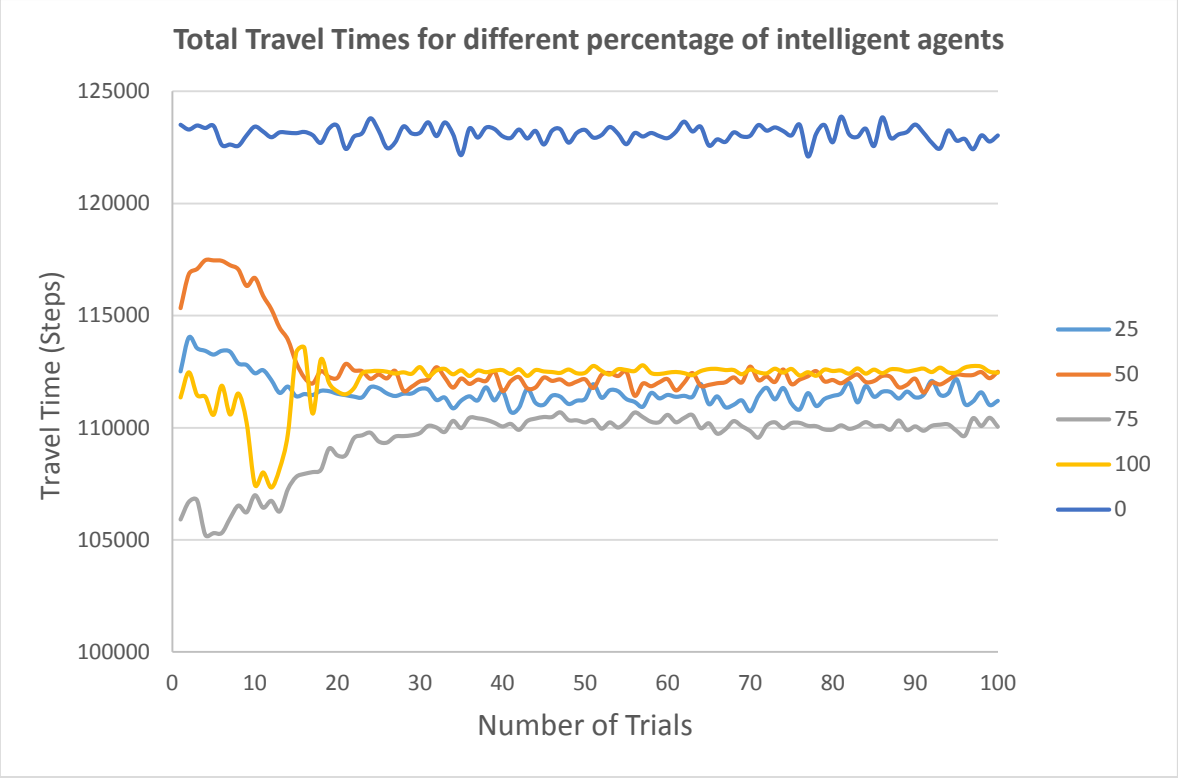
If $d_y = 0$

$$Posssible\ Actions(state) = \{R$$

Considering the constraints that an agent cannot step outside the network. Also, an agent cannot pass its destination vertically. Since the objective is to come up with optimal quickest path, I manipulated possible actions in a way that an agent ends up in its destination for sure, and spends the learning potential for finding quickest path. Next section represents the results of this part applied on a simple three by three network.

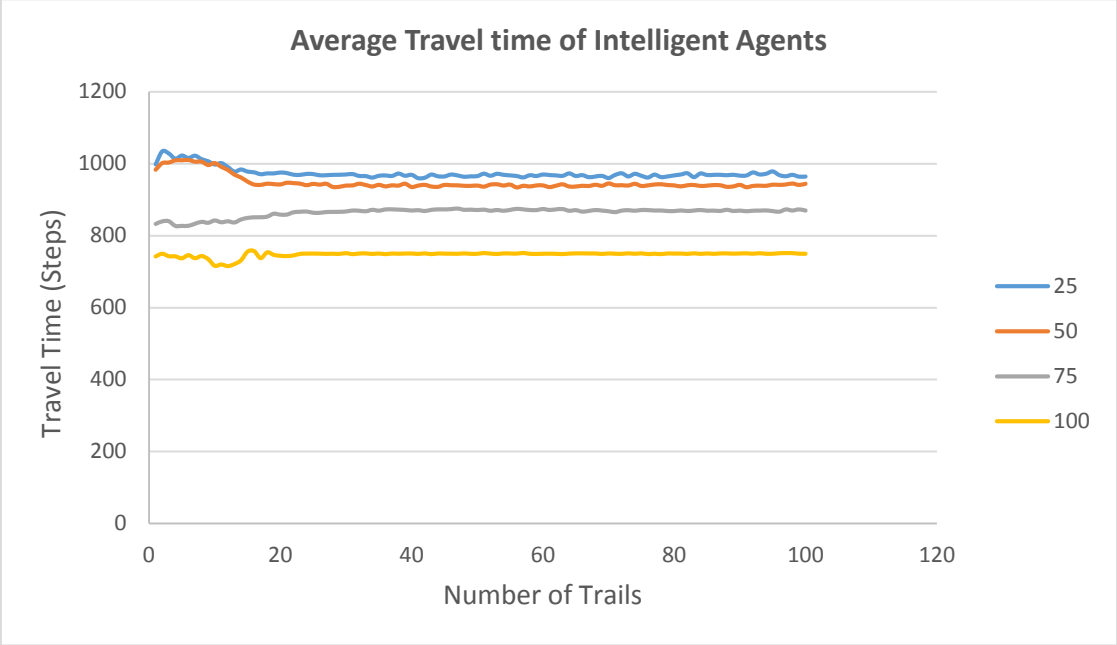
Results

The result of this part is very interesting to me. The next graph shows the evolution of total travel time

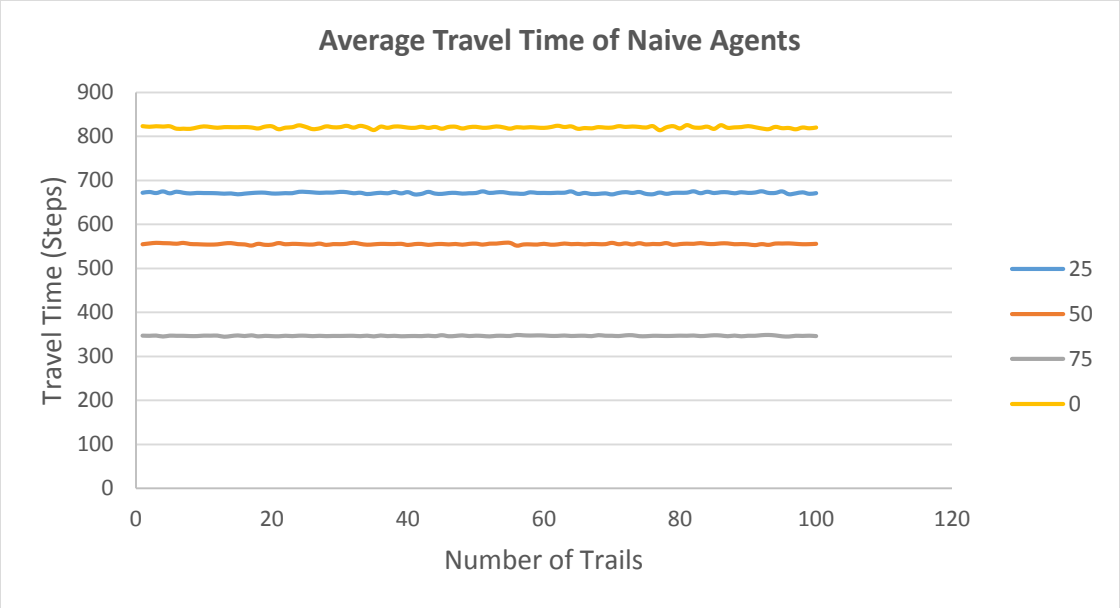


Of the all agents of the network (Naïve and Intelligent) for different percentages of the intelligent agents, the final converged total travel times is minimum when 75% of the agents are intelligent, not all of them. This suggests that as the number of intelligent agents increase, convergence to the optimal solution gets harder or even impossible. This makes sense since all of the agents are trying to optimize their utility function, and if their benefits contradict with the others, reaching optimal solution is not feasible. In this case, having 100% of intelligent agent is worse than all other three different scenarios. All in all, the bright side is that having intelligent agents improves the performance of the system, compared to not having any.

The next graph exhibits the average travel time of intelligent agents, which possesses interesting information as well. If you take a look at learning curve of the intelligent agents for 75% and 100%, you will notice that the performance is deteriorating over time, instead of improving. On the other hand, the performance of systems with 25% or 50 % intelligent agents improves over time. Therefore, I propose there would be a threshold between 50% and 75%, above which the systems deteriorate, and below which systems improve over time.



The next graph also reveals that how having intelligent agents in the system affects the average travel times of naïve agents. As you can see, as the percentage of intelligent agent increases, the average travel time of naïve agents decreases.



Conclusion

This project consists of two major sections. First, apply a Q-learning algorithm with reverse order Q Matrix updates to reach the optimal quickest path within a grid network from the top left corner to the bottom right corner, during which different coefficients have been evaluated an analyzed based on their impact on the algorithm's performance. Second, implement learning algorithm to the multi-agent systems with different percentage of intelligent agents.

The results revealed that as the percentage of intelligent agent increases, it is more difficult to converge to the optimal solutions. It can be explained that when all of the agents are trying to optimize their utility function, and if their benefits contradict with the others, reaching optimal solution is not feasible. In addition, this study found that the final converged total travel times reach minimum when 75% of the agents are intelligent. Due to the computation and time limit, the critical value of under what percentage of intelligent agents, the travel time achieves minimum is uncertain at this point. However, the results of part II suggest that the scope of the threshold would lie between 50% and 75%, above which the systems deteriorate, while below which systems improve over time. This further validates that increasing percentage of intelligent agents does not necessarily improves the performance of the system.