Northeastern University
College of Engineering

Project Work for the class IE7295 Reinforcement Learning under Prof. M. Dehghani

Submitted by Group 1:

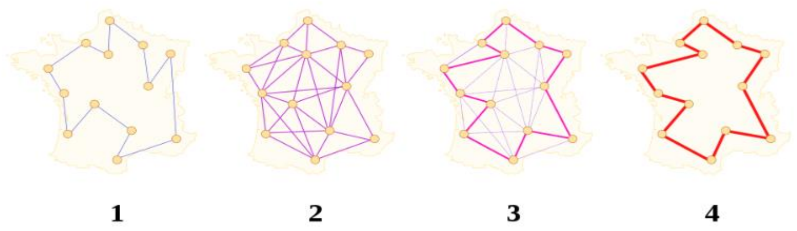
Pulkit Saharan and Shivani Naik

What is Travelling Salesman Problem?

Travelling Salesman Problem is one of the most conventional problems seen in the space of reinforcement learning. It is an optimization problem wherein we must find the shortest possible route to visit different places/cities. We need to determine the shortest path that visits each city precisely once, returns to the starting city, and is given a list of cities and the distances between each pair of cities. This corresponds to the properties of a NP-hard problem and hence is categorized as such. When a variety of constraints, such as multiple vehicle fleets, time window constraints, and so on, are added to the travelling salesman problem, it becomes even more difficult to solve. Hence, a combination of the variants with heuristics is used by delivery systems in practice.

Commented [SN1]: i don't think we have done this

Commented [PS2R1]: mne headings change krdi, i think it is making sense now



Problem Statement

In this work, we have formulated the TSP as an RL (Reinforcement Learning) problem by identifying the states, actions, and rewards spaces. We have implemented Q-learning and SARSA algorithms based on these space definitions and trained these agents to solve the optimization problem and have conducted experiments to analyze the effect of hyperparameters of these algorithms. For additional constraints, we have introduced a traffic area constraint in the vanilla TSP.

Literature Review

Since, the inception of the Reinforcement learning space Travelling Salesmen Problem is one of the classic examples of real-world scenarios in which RL can be implemented and the results would create a better environment for salespeople to maximize his expected rewards. In today’s world, where we can get everything delivered to us by the touch of our fingertips, it is imperative that we can solve this problem as it will lead to increased efficiency in the entire delivery process which has become an integral part of our lives.

There are two basic types of TSP:

1. Symmetric TSP (STSP): Distance between A to B is same as the distance between B to A
2. Asymmetric TSP (ATSP): Distance between A to B is different from distance between B to A. This can happen when there are one-way roads between A to B, thus we always have a different minimum distance in one direction

One of the most basic approaches for solving this problem was the Nearest-Neighbour algorithm. It is the simplest heuristic algorithm used to solve TSP. The algorithm can be summarized in the following steps. First, select a random city n and set it as the starting city n0. Then, find the nearest unvisited city and go there, consequently, mark the current city as visited. Then, again check if there are any unvisited cities? If yes, go to (2) otherwise, lastly, return to the starting city.

One of the other approaches which we found during our research into the problem was the usage of greedy and near-greedy approaches to solve the problem. In this approach, the first step is to introduce the function that defines an ε-greedy action for a given state. So, with probability 1-ε and if we have already defined a greedy action for that state, it selects the greedy action and a random one otherwise. This function specifies the starting and ending point as **'X'** and it initializes an empty list called history that will contain the visited states. Moreover, it creates a list that contains all the cities except ***'X'*** since it is the first one visited. After that, the agent enters the while loop and selects the next actions until he visits all the cities and returns to Turin. In this way, we are building an episode. The actions are chosen using the near-greedy function and appends the incremental value to the return of each visited state. It computes the state-value for all of them by averaging their returns. Lastly, it defines the action-value for the states that the agent visited in this episode by updating the value in the dictionary Q which is returned.

One of the most interesting methods or a benchmark algorithm which we encountered in our research is Concorde, which is a computer code for the symmetric traveling salesman problem (TSP) and some related network optimization problems. The Concorde callable library includes over 700 functions permitting users to create specialized codes for TSP-like problems. All Concorde functions are thread-safe for programming in shared-memory parallel environments. The code is written in the ANSI C programming language, and it is available for academic research use.

Methodology

The methodology proposed for this project is divided into two parts. The Traveling Salesman Structure, first Establishing the goal, agent, states, actions, and rewards will help define the problem in terms of reinforcement learning. The second is a problem-solving algorithm that identifies the optimal policy to determine the shortest path.

Problem Formulation: Travelling Salesman Problem to Reinforcement Learning

The agent will be taught how to find a route that minimizes distance using the RL environment. A set of states, actions, and rewards make up the RL environment for the TSP in this case.

Goal: Finding the shortest path to travel between all the cities

Environment: Created our own environment to define set of states, actions, and rewards

States: Cities(nodes) the agent (traveling salesperson) visited to complete the route. Nodes/cities have indices and are represented by (x,y) coordinates. A list of visited nodes is also part of the state.

Actions: Movement to another city that has not yet been visited

Reward: Negative Euclidean Distance between the current city and the next city. The greater the distance between the cities, smaller the reward will be. The agent will try to maximize this reward.

For a variation of including traffic zones, the base reward of negative Euclidean distance is increased by the additional reward of negative Euclidean distance times traffic intensity if the route crosses through a busy location

*Base Reward = -1*distance between two cities*

*Additional Reward = Base Reward * Traffic Intensity*

Episode Termination: An episode terminates when all the states are visited

RL Algorithms

In Reinforcement Learning, a variety of algorithms are available to help an agent learn and accomplish the goal by determining the optimal policy. In this project, we have implemented two different algorithms - SARSA and Q-learning. These are common RL algorithms and based on temporal difference learning (TD), that is, updates do not need to refer to real-time intervals, but to successive decision-making steps.

1. Q-Learning Algorithm

Q-Learning is an off-policy reinforcement learning algorithm that tries to find the best action for the current state. The Q-Learning feature is considered off-policy because it learns from current off-policy actions such as: Random Action. Therefore, no policy is required. More specifically, Q-Learning tries to learn policies that maximize the overall reward. The ‘q’ in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

The Q-Learning equation is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

2. SARSA Algorithm

The SARSA algorithm is a slight variation of the Q-learning algorithm. There are two types of policies for learning agents in reinforcement learning algorithms.

On Policy: The learning agent learns the value function according to the current action derived from the currently used policy.

Off Policy: A learning agent learns a value function according to actions derived from another policy.

The Q-learning method is an off-policy method that uses a greedy approach to learn Q values. On the other hand, the SARSA technique, which is on-policy, is policy-compliant and uses the actions taken by the current policy to learn the Q-value.

SARSA is an RL on-policy TD Control method that, in accordance with the following equation, depends on the subsequent action (at+1) specified by the policy π (s) to update the learning matrix:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

where ***S*** is a state and ***A*** is an action at the current instant (t), respectively;

St+1 is state and ***At+1*** is action at time (t+1)

Q (St, At) is the value at time t for state-action pair ***(S, A)***

α is the learning rate; ***γ*** is the discount factor.

Results

In this project, multiple simulations have been implemented based on the number of cities, learning algorithms and environment conditions. This project includes two environment conditions –

- Vanilla TSP: When the environment is free
- TSP with Traffic Zone: When a traffic zone is introduced into the environment it slows down traffic

We have created two classes of agents and implemented Q-Learning and SARSA on 5, 50, 100 and 500 number of cities to generate optimal route for every combination.

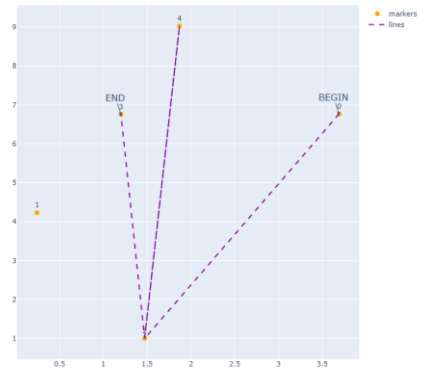
Vanilla TSP (Without Constraints)

When we talk about a "vanilla TSP," we are talking about a straightforward TSP where there are no restrictions and the flow of traffic is smooth, and where the payout depends just on the distance between two cities.

*Reward = -1 * Distance between two cities*

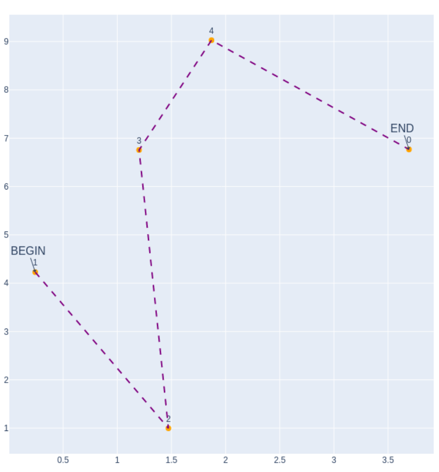
1. 5 Cities

We started with five cities and generated an environment with a starting and ending point. Below graph shows an iteration of randomly visited cities where the orange dots represent the location of a city, and the purple dashed lines represent the path taken between the two cities.

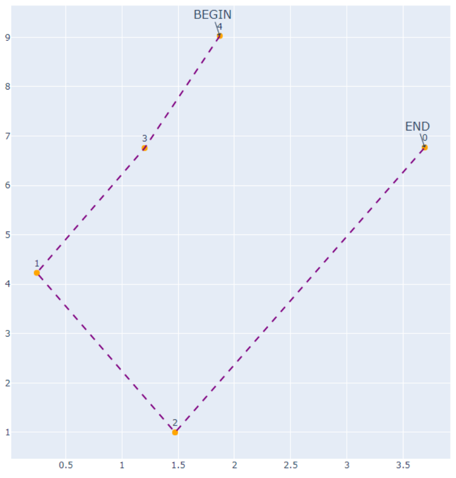


Q-Learning Agent

Below graph shows the before training and after training results.



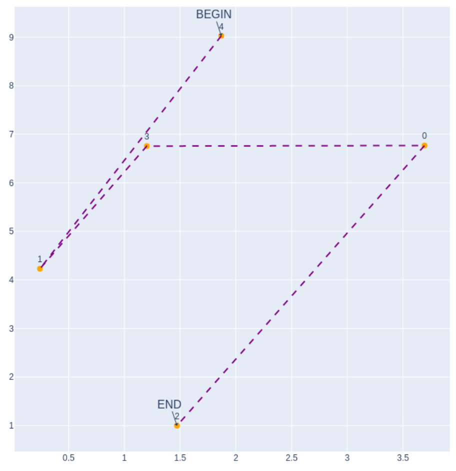
Before Training



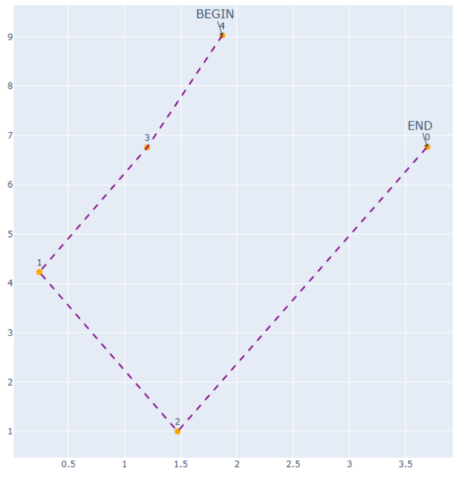
After Training

GIF of training: [Gif]

SARSA Agent



Before Training

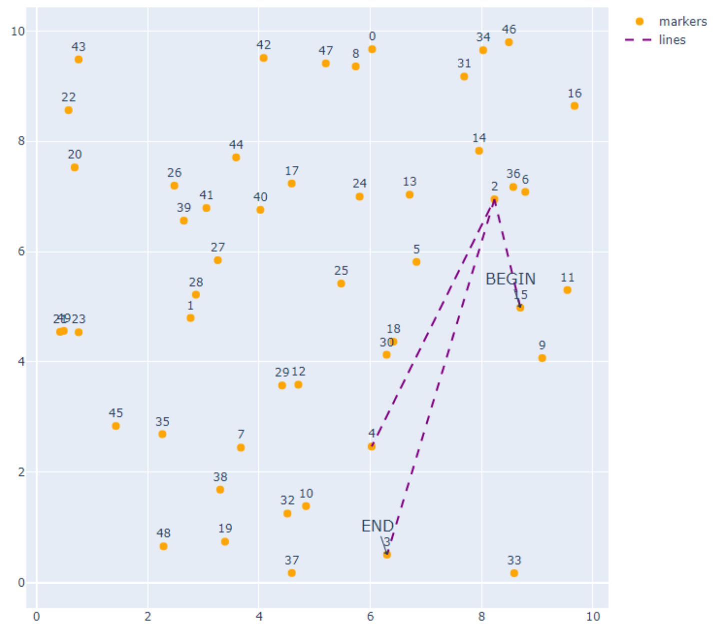


After Training

GIF of training: [Gif]

2. 50 Cities

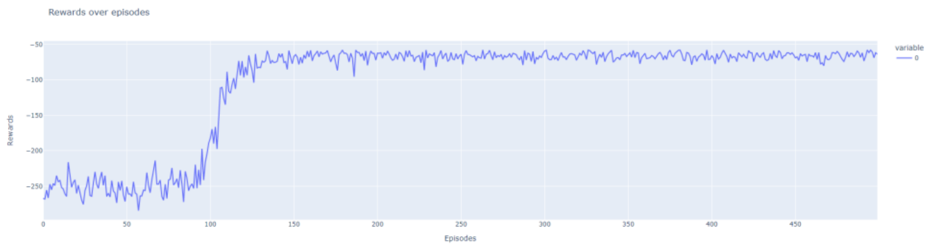
We implemented both the learning agents in fifty cities. The graph below shows randomly visited cities in an environment with fifty cities given a starting and ending point.



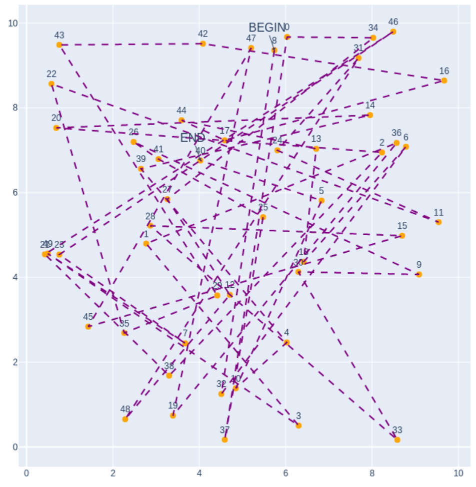
Q-Learning Agent

The graphs below demonstrate the outcomes of applying Q-Learning to a vanilla TSP environment in fifty cities. The line graph below, which displays the movement of reward in relation to the number of episodes, makes it evident that the agent was learning the various routes between cities during the initial phase, which lasted up until 100 episodes. In the initial phase, the agent used random paths with an epsilon decay value of 0.999.

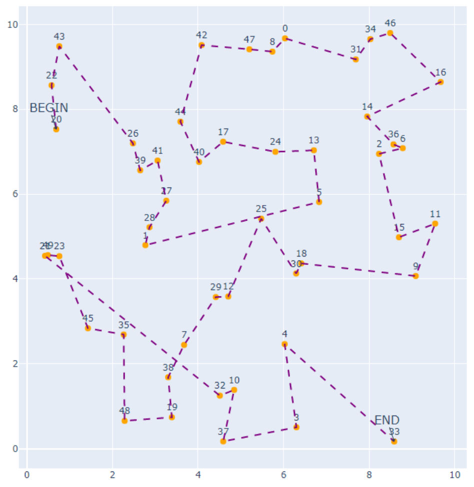
As epsilon decreased due to the epsilon decay component in the later phase, the agent began to make use of learning, acting less randomly and more in accordance with Q-values. As the number of episodes exceeds 100, we can see that the reward values rise sharply.



The path before and after training is depicted in the two graphs below. The route is shown on the left-hand graph before training, and it is obvious that it is not the best one because the agent repeatedly travels the same path, increasing travel distance and lowering the reward. The route that is best for achieving the lowest distance and most reward after training is depicted on the right-hand graph. In every experiment, the algorithm quickly finds a path that is reasonable. After examining several possibilities, it not only offers one path but also variations of the recommended course of action, which might already be intriguing to uncover alternatives.



Before Training



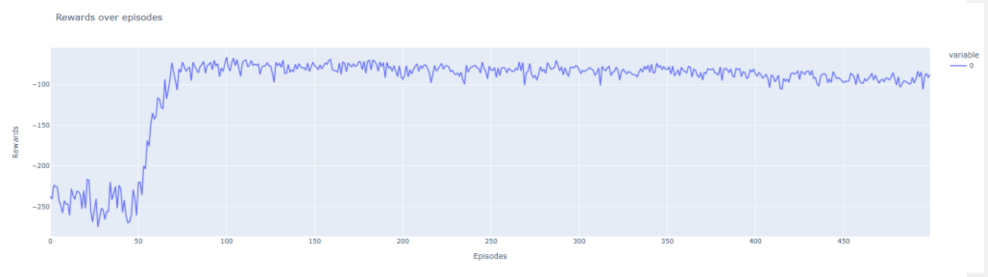
After Training

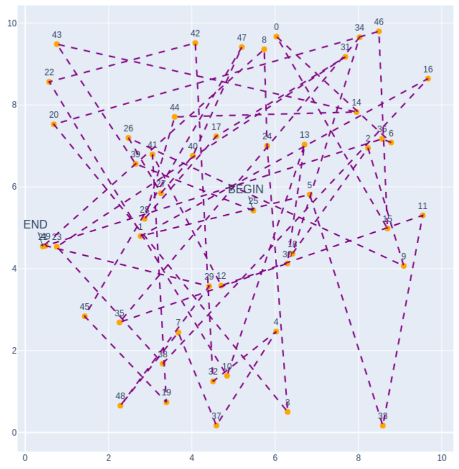
GIF of training: Gif

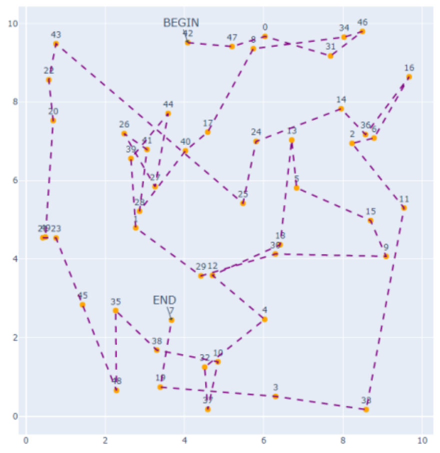
SARSA Agent

The outcomes of our training of SARSA agents in fifty cities are shown here. In terms of incentives, the SARSA agent's outcomes are quite like those from Q-Learning. There can be more than one optimal path with the same reward, hence the SARSA agent's optimal path differs from the Q-Learning one.

The graph below demonstrates how reward rises as the number of episodes rises and the stage transitions from exploration to exploitation.





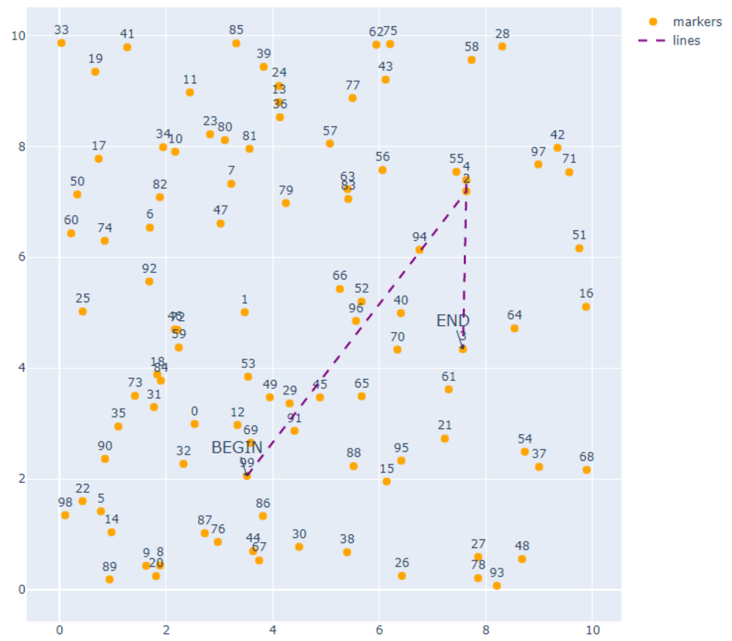


Before Training After Training

GIF of training: [Gif]

3. 100 Cities

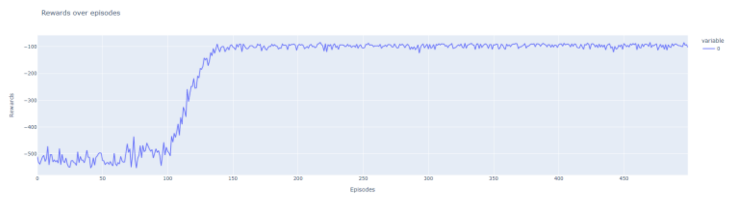
We implemented both the learning agents in 100 cities. The graph below shows randomly visited cities in an environment with 100 cities given a starting and ending point.



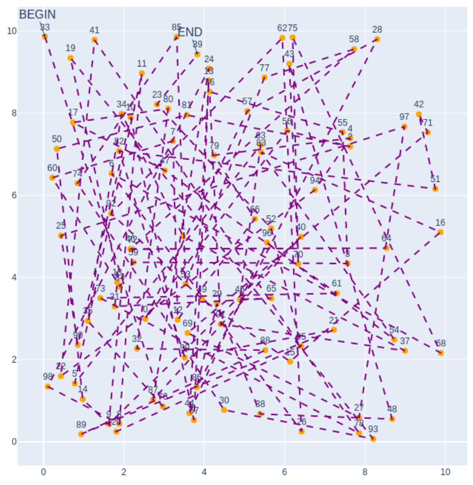
Q-Learning Agent

The graphs below show the results of implementing Q-Learning in 100 cities with a standard TSP environment. The outcomes in the case of 100 cities are fairly comparable to those in the case of 50 cities. In the initial phase, which lasted up to 100 episodes, the agent was clearly learning the various routes between cities as shown by the line graph below, which shows the movement of reward in response to the number of episodes. The agent used random pathways with an epsilon decay value of 0.999 throughout the initial phase.

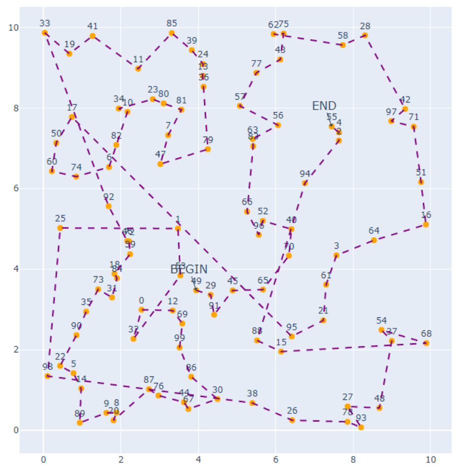
The agent started to employ learning when epsilon dropped due to the epsilon decay component in the later phase, operating less randomly and more in accordance with Q-values.



Below graph shows the before training and after training routes in 100 cities environment.



Before Training

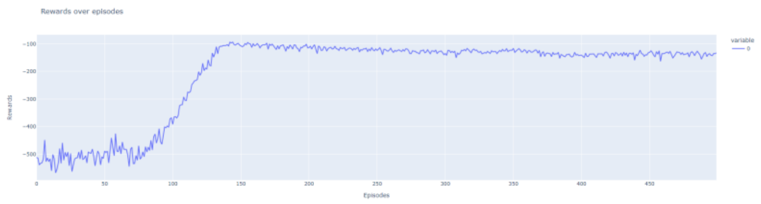


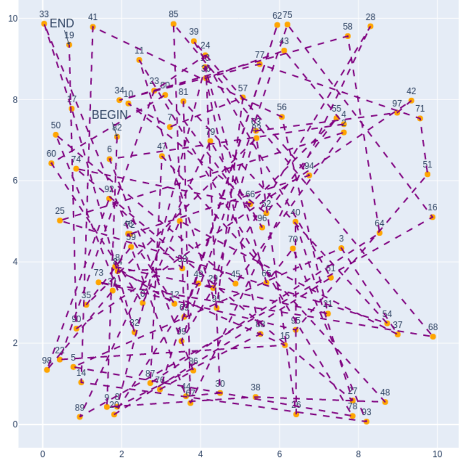
After Training

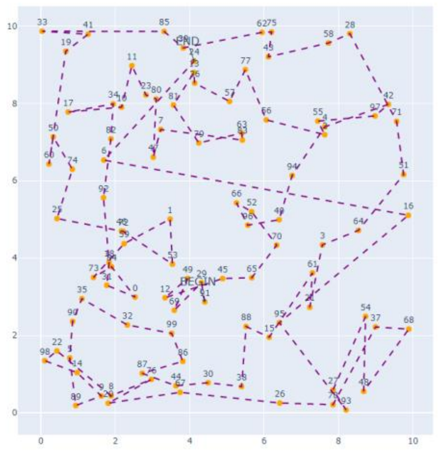
Gif

SARSA Agent

The results in the case of 100 cities are also similar for SARSA agent and Q-Learning agent as given below.





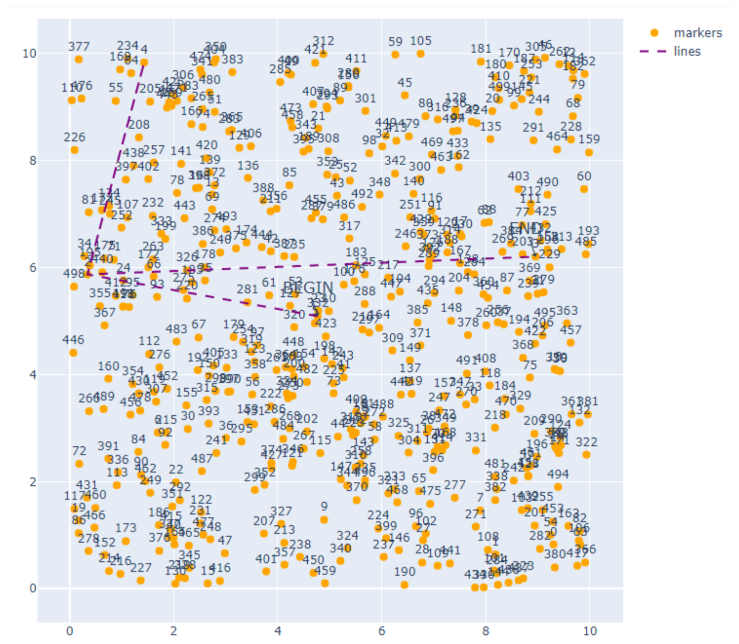


Before Training After Training

Gif

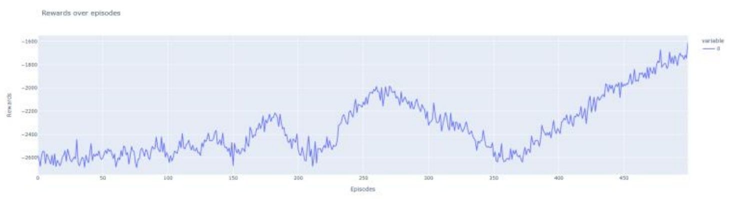
4. 500 Cities

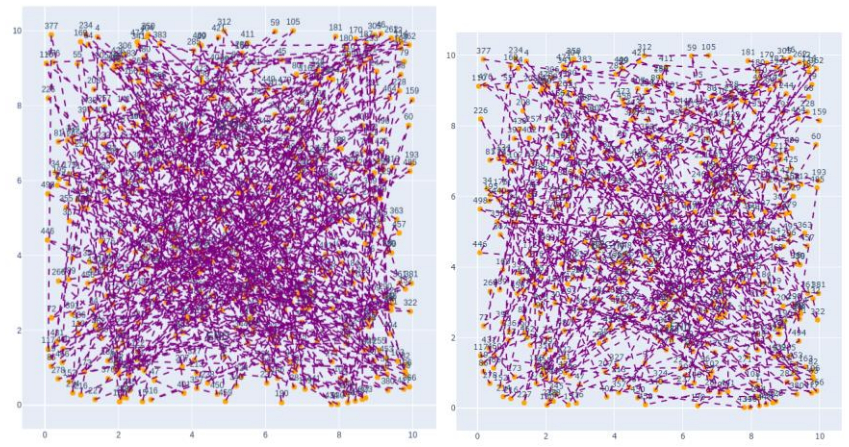
We implemented both the learning agents in 500 cities. The graph below shows randomly visited cities in an environment with fifty cities given a starting and ending point.



Q-Learning Agent

The graphs below show the results of implementing Q-Learning in 500 cities with a standard TSP environment. The line graph below, which shows how reward changes over time in response to the number of episodes, demonstrates that there was initially a lot of exploration followed by an increase in reward after 350 episodes. As the environment is complicated, has a greater number of cities, and has unlimited paths, the agent in this scenario took a significant number of episodes to investigate before moving towards higher rewards with degraded epsilon. The graph below and prior observations make it clear that agents need more episodes to learn a complicated environment.



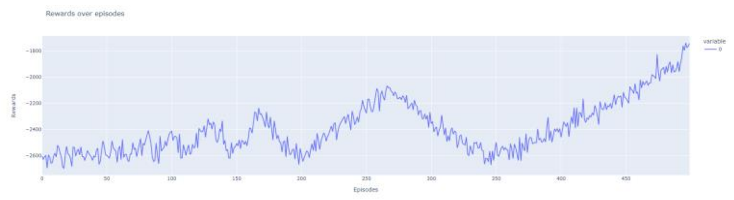


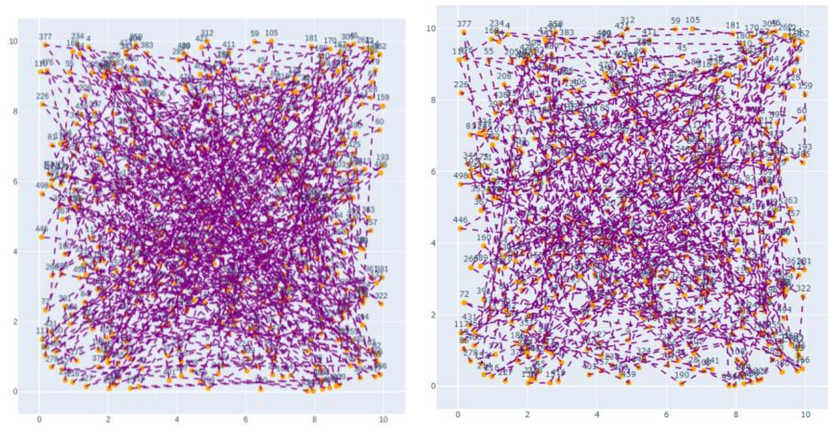
Before Training After Training

Gif

SARSA Agent

We can see comparable results for SARSA as well.





Before Training After Training

Gif

With Traffic Constraint

We have the results for the vanilla TSP in the section above. We have changed the setting to reflect a more complicated and real-world scenario. For stress testing our RL algorithms and evaluating their performance in a challenging setting, we have incorporated a traffic zone into our environment where routes will be slower if chosen depending on the volume of traffic. We determined the point where the routes crossed the traffic zone, measured the length of the segment that was through the traffic area, and then included in the traffic intensity factor to account for the complexity of our environment.

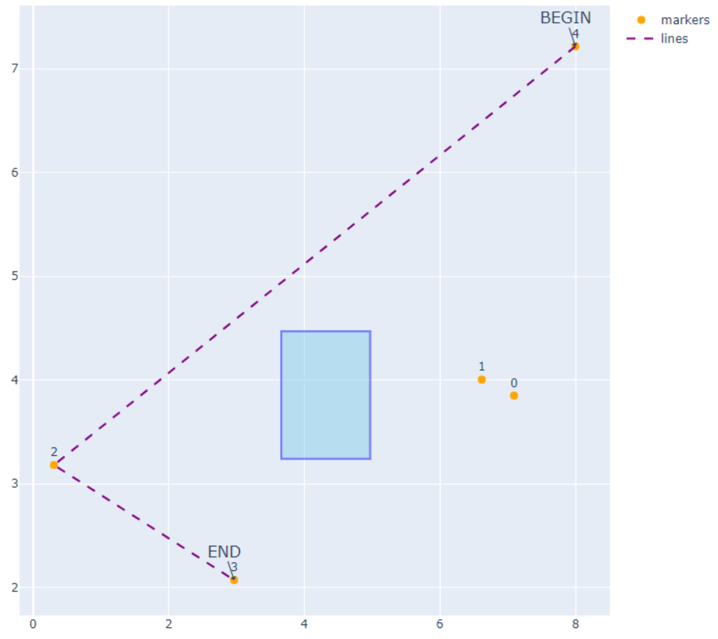
We must decide the rewards in this scenario after establishing the traffic zone, crossing locations, and weighted segments leaving the traffic zone. An additional negative reward will be added to our base reward in a circumstance like this. Now that only the reward function has been altered, the agent will continue to study its surroundings in the same manner and optimize its route.

*Base Reward = -1*distance between two cities*

*Additional Reward = Base Reward * Traffic Intensity*

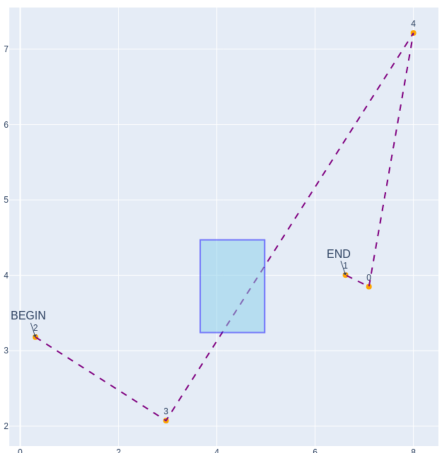
1. 5 Cities

We started with five cities and generated an environment with a starting and ending point. Below graph shows an iteration of randomly visited cities where the orange dots represent the location of a city, and the purple dashed lines represent the path taken between the two cities. The blue box in the graph below shows the traffic area.

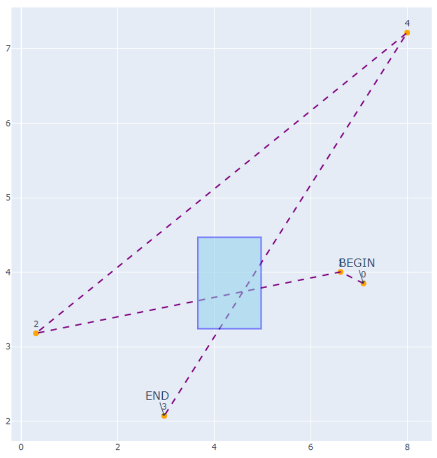


Q-Learning Agent

Below graph shows the before training and after training results.



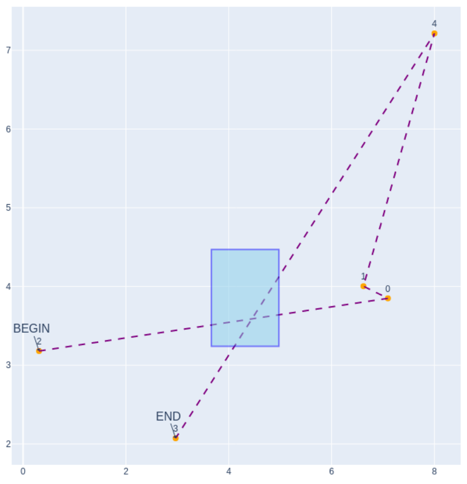
Before Training



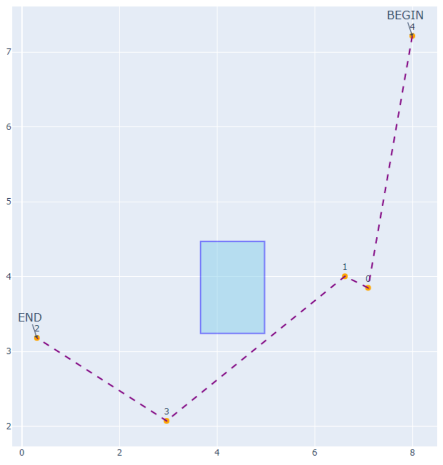
After Training

Gif

SARSA Agent



Before Training

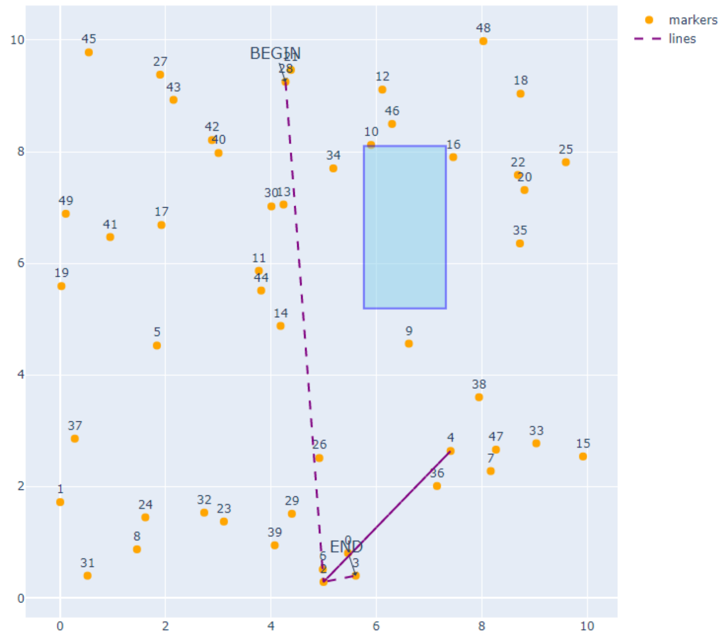


After Training

Gif

2. 50 Cities

We implemented both the learning agents in fifty cities. The graph below shows randomly visited cities in an environment with fifty cities given a starting and ending point.

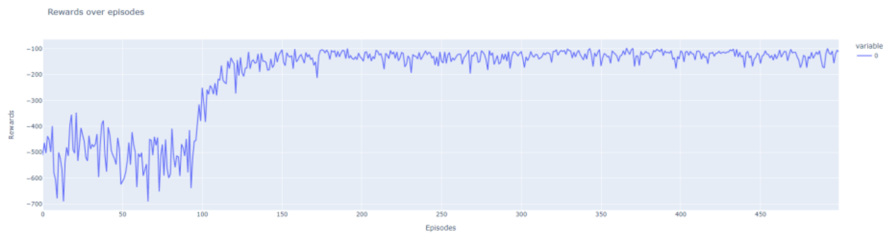


Q-Learning Agent

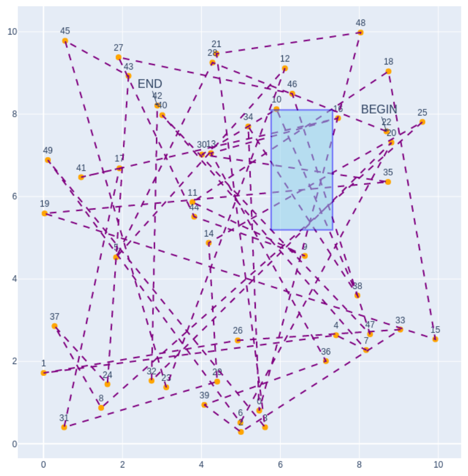
The results of implementing Q-Learning in a limited TSP setting across fifty cities are shown in the graphs below. In the initial phase, which lasted up to 100 episodes, the agent was clearly learning the various routes between cities as shown by the line graph below, which shows the movement of reward in response to the number of episodes. The agent used random pathways with an epsilon decay value of 0.999 throughout the initial phase.

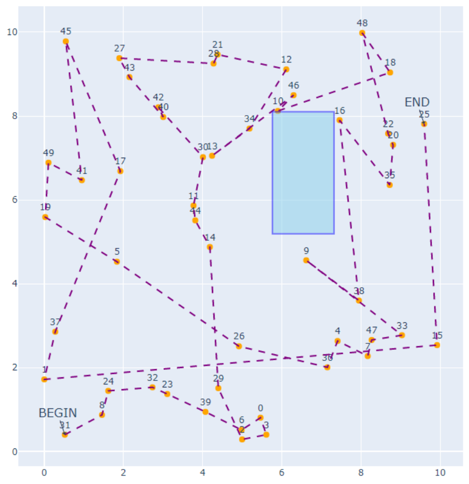
The agent started to employ learning when epsilon dropped due to the epsilon decay component in the later phase, operating less randomly and more in accordance with Q-values. We can see that the prize values increase significantly as the number of episodes exceeds 100.

The only discernible difference between this behavior and that of the Vanilla TSP environment is the reward value. Due to the additional negative reward value given to the base reward, which will reduce total awards, the reward value after the traffic zone's introduction is somewhat lower than without traffic.



The two graphs below show the course both before and after training. Prior to training, the route is displayed on the left-hand graph, and it is clear that it is not the ideal one because the agent frequently follows the same path, increasing trip distance and ultimately diminishing the reward. The right-hand graph shows the route that offers the greatest reward with the least amount of travel. The algorithm immediately identifies a plausible path in every experiment. It not only presents one way after looking at numerous options, but also different directions that may be taken, which could already be interesting to find alternatives.





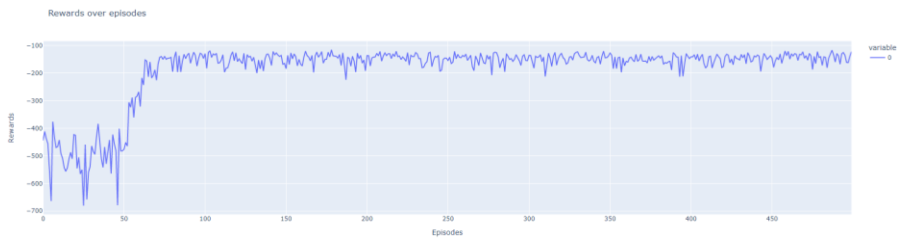
Before Training   After Training

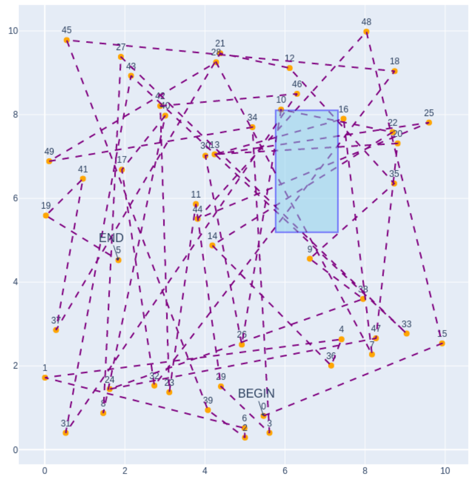
Gif

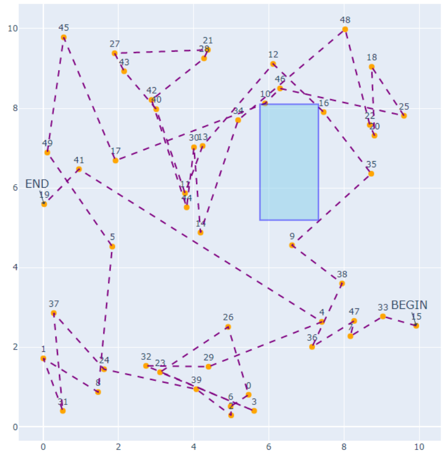
SARSA Agent

Here are the results of our training SARSA agents in fifty cities. The SARSA agent's results in terms of rewards are very comparable to those from Q-Learning. The SARSA agent's optimal path is different from the Q-Learning agent's because there can be more than one optimal path with the same reward.

The graph below shows how reward increases as episode count increases and stage changes from exploration to exploitation.







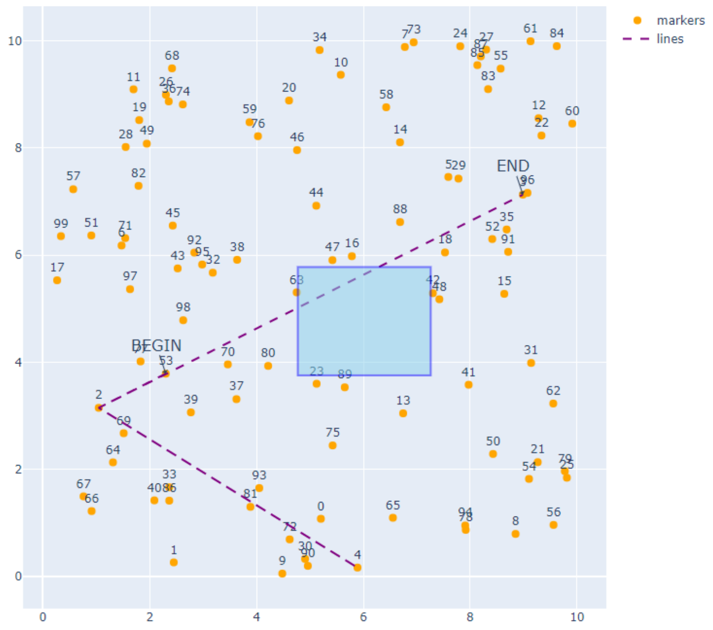
Before Training

After Training

Gif

3. 100 Cities

In 100 cities, both learning agents were used. In a setting with 50 cities and a starting and ending point, the graph below depicts randomly visited cities.

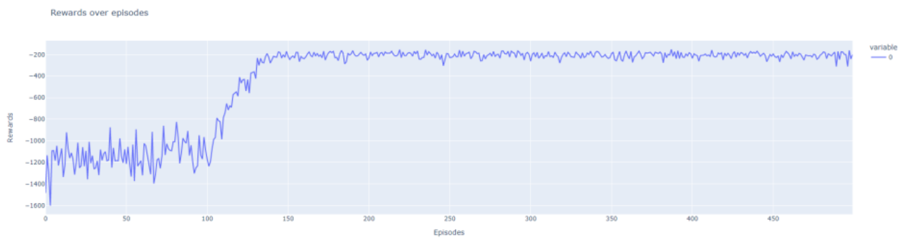


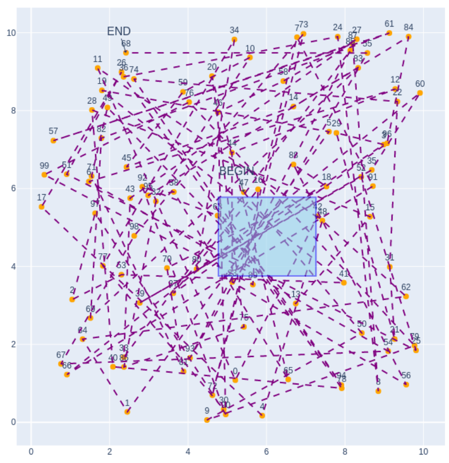
Q-Learning Agent

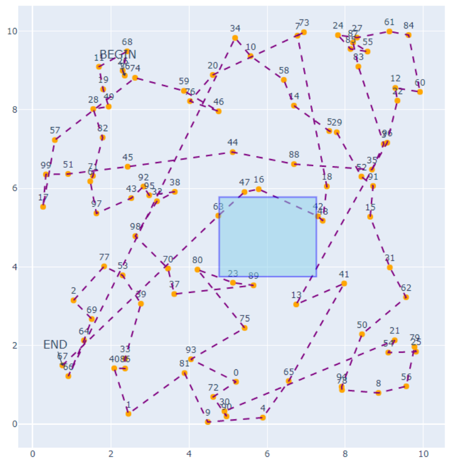
The outcomes of applying Q-Learning in 100 cities using a constrained TSP environment are displayed in the graphs below. The results for 100 cities are equivalent to those for 50 cities. The line graph below, which displays the movement of reward in relation to the number of episodes, demonstrates how the agent was obviously learning the various routes between cities throughout the initial phase, which lasted up to 100 episodes. During the initial phase, the agent used random paths with an epsilon decay value of 0.999.

When epsilon decreased due to the epsilon decay component in the later phase, the agent began to use learning, functioning less randomly and more in line with Q-values.

As we can see in this instance as well, the reward value is the obvious distinction between this behavior and that of the Vanilla TSP environment. The reward value following the introduction of the traffic zone is lower than without traffic due to the additional negative reward value added to the base reward, which will lead to a decrease in total rewards.





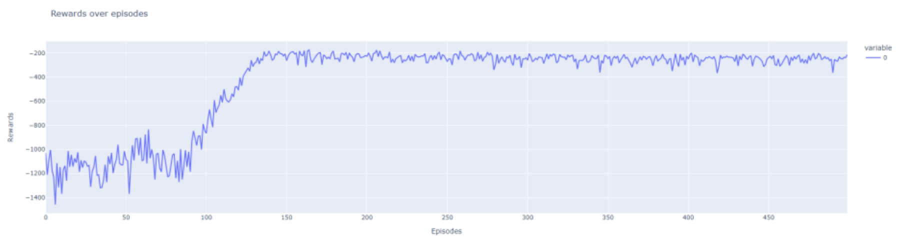


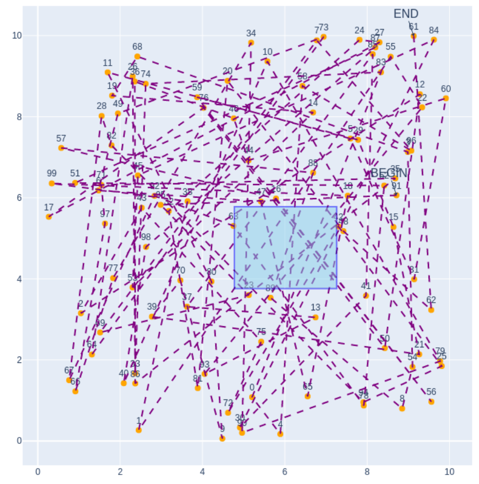
Before Training After Training

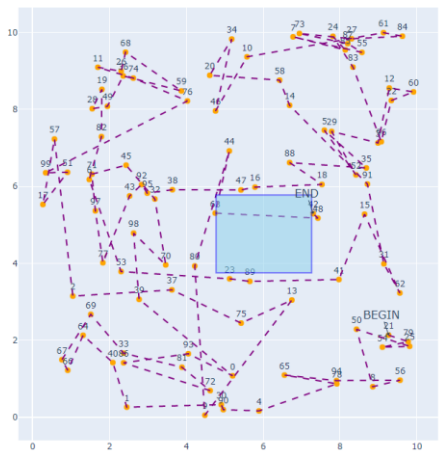
[Gif]

SARSA Agent

When it comes to initial exploration and convergence toward the ideal reward value, SARSA agent behaves similarly to the Q-Learning agent.







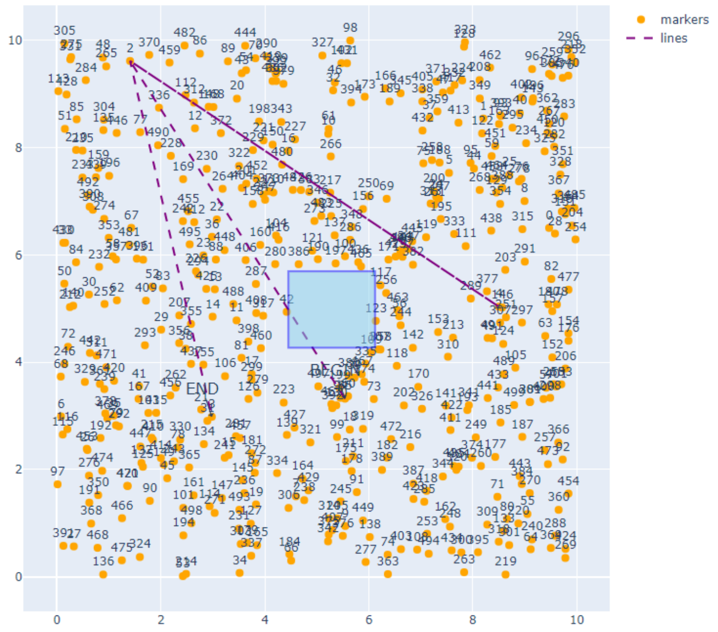
Before Training

After Training

Gif

4. 500 Cities

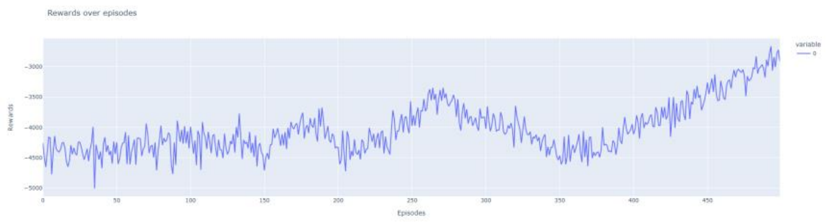
We implemented both the learning agents in 500 cities. The graph below shows randomly visited cities in an environment with fifty cities given a starting and ending point.

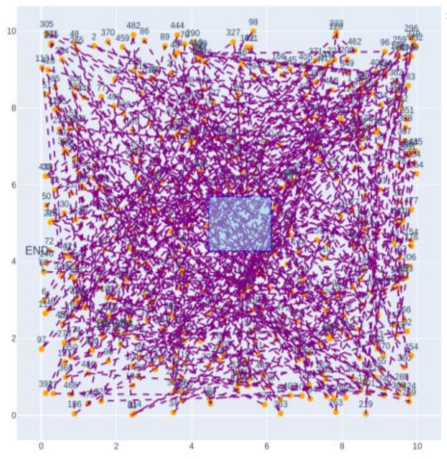


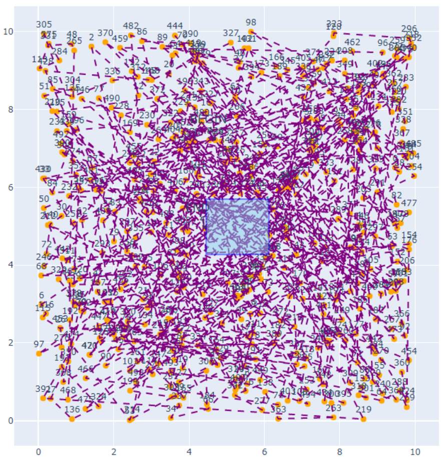
Q-Learning Agent

The outcomes of implementing Q-Learning in 500 cities with a restricted TSP environment are displayed in the graphs below. There was originally a lot of exploration followed by an increase in reward after 350 episodes, as seen by the line graph below, which depicts how reward changes over time in relation to the number of episodes. The agent in this case takes a substantial number of episodes to investigate before heading towards higher rewards with worse epsilon since the environment is more difficult, includes a greater number of cities, and has limitless pathways. It is evident from the graph below and earlier findings that agents require more episodes to fully understand a complex environment.

The clear distinction between this behavior and that of the Vanilla TSP environment, as we can observe in this instance as well, is the reward value. Due to the additional negative reward value added to the base reward, which will result in a drop in total rewards, the reward value after the traffic zone is much lower than it would have been without traffic.







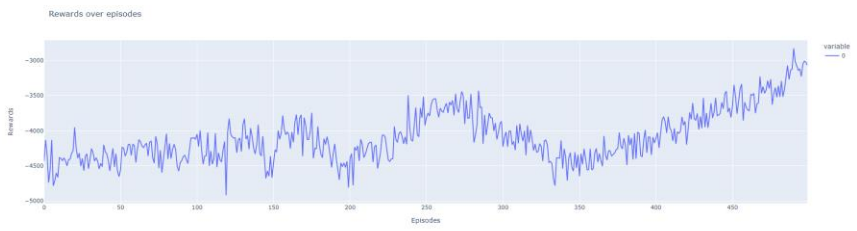
Before Training

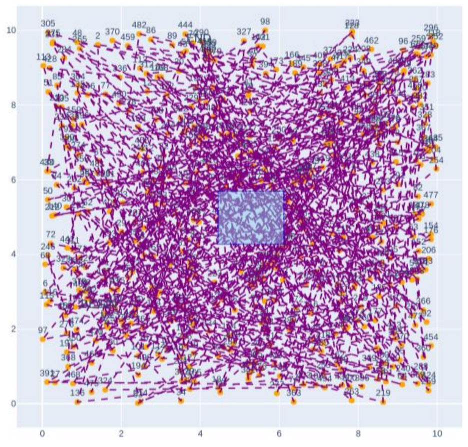
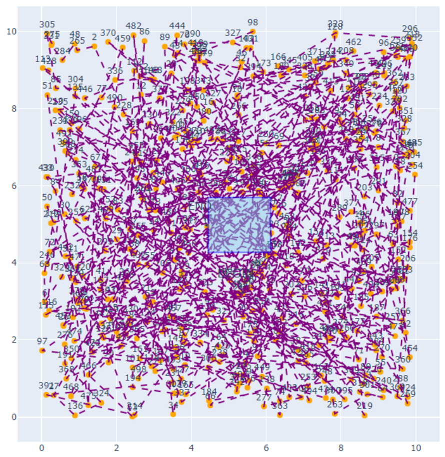
After Training

Gif

SARSA Agent

SARSA agent is behaving like the Q-Learning agent by exploring in the initial episodes and converging towards the optimal reward value.



Before Training After Training

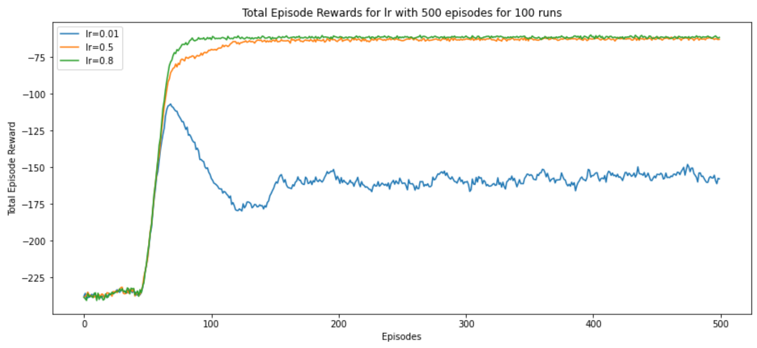
Gif

Experiments with Hyperparameters

There are three important hyperparameters, used in both the algorithms, which control the training of the algorithm and significantly impact the performance of the model which is being trained. In this project, analysis has been done to compare the performance of Q-Learning algorithm with 50 cities in case of different values of the hyperparameters.

1. Learning Rate

Learning rate regulates learning pace; it establishes how quickly you pick up new states of experience. In the experiment, 500 episodes with 100 runs each were used to evaluate the performance of the Q-Learning algorithm using three different learning rate settings (0.01,0.5,0.8). The graph below shows that during the first episodes while the agent was exploring, the reward was very low and equal for all values of learning rate. All three learning rate values saw their rewards rise sharply as the number of episodes rose, however those with learning rates of 0.5 and 0.8 perform better than those with an extremely low learning rate of 0.01.



2. Epsilon

Epsilon is the probability of taking random action in epsilon-greedy policy.

Exploration: When an agent investigates, it may not always behave in accordance with its knowledge; instead, it may explore many choices based on an exploration strategy.

Exploitation: An agent can act greedily at any time based on the knowledge it knows to maximize the overall benefits it expects to receive.

Exploitation has a significant disadvantage. The agent only acts based on its limited and incomplete understanding of the environment. If an agent's contact with the environment has barely started, this is particularly problematic. It is exceedingly improbable that an agent will learn the best behavior in the environment if it merely acts greedily based on its poor understanding of the environment.

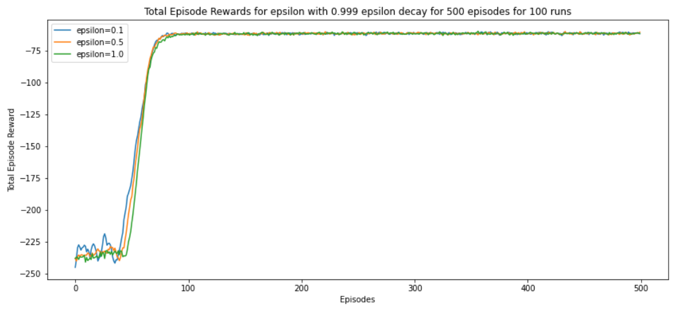
During RL training, the Epsilon-greedy policy method can be used to strike a balance between exploration and exploitation. For instance, an epsilon value of 0.4 indicates that the output action is picked at random from the action space with a probability of 0.4, while an epsilon value of 0.6 indicates that the output action is chosen greedily based on the maximum Q value.

Epsilon Decay

It is also crucial to remember that when an agent is dealing with an environment for which it lacks sufficient knowledge, exploration becomes even more crucial. Allowing an agent to utilize its knowledge makes more sense if it has the knowledge required to interact with the environment appropriately. Thus, for an agent to eventually learn and behave ideally, the value of epsilon should decrease over its existence. This is sometimes accomplished by multiplying Epsilon by a real number smaller than 1 for each episode. For instance, Epsilon = Epsilon*0.9, With more occurrences, this would cause epsilon to gradually degrade. This small number is known as the epsilon decay value.

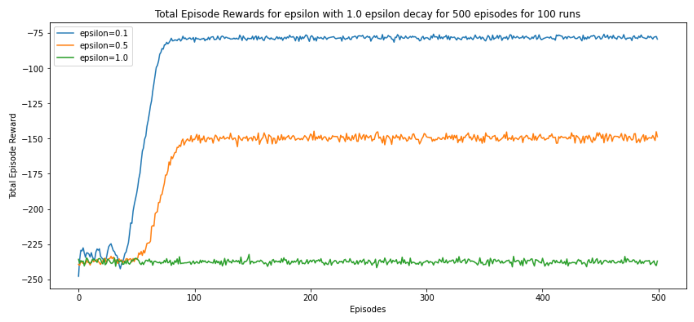
a. Epsilon Decay = 0.999

Three distinct epsilon values, 0.1, 0.5, and 1, were considered in the experiment, with an epsilon decay value of 0.999. The graph below shows that for all three values of epsilon, the agent is in the exploration stage during the first phase when the number of episodes is less than 50. Epsilon value decreases by a factor of epsilon decay as the number of events rises and learning advances. As a result, the agent uses its prior knowledge to take advantage of the surroundings and gain larger reward value. All three epsilon values will decay and converge to similar rewards.



b. Epsilon Decay =1 (No Decay)

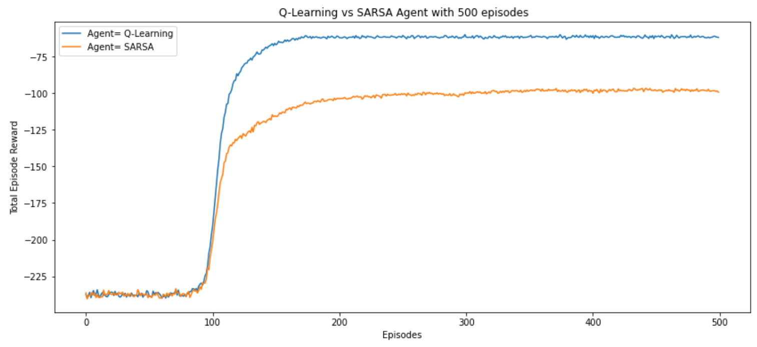
With the value of decay set to 1, there should be no decay in this experiment and the value of epsilon will remain constant throughout all episodes. In the graph below, we can observe that all three graphs in the early phase depict the exploration stage with a little bit greater volatility in rewards for the agent with epsilon equal to 0.1 since it follows a greedy policy with very little likelihood of doing random action. The incentive for lower epsilon values has increased after the exploration phase as the agent learns and behaves ideally. An agent with an epsilon value of 1 remained in the exploration stage and received lowest rewards because it constantly chooses actions at random.



Q-Learning vs SARSA

QL and SARSA are both excellent initial approaches for reinforcement learning problems. In a limited space, both strategies are effective (or a discretized continuous environment). While SARSA learns a "near" optimal policy, QL directly learns the optimal strategy. In contrast to SARSA, QL is a more aggressive agent.

The performance of the Sarsa and Q-Learning methods for 50 cities and 500 episodes of 100 runs each with an epsilon decay value of 0.999 is shown in the graph below. We can observe that over time, Q-Learning outperforms SARSA in terms of performance. Q-learning and SARSA were in the exploration stage and receiving less reward in the early portion of the plot when there were fewer than 100 episodes. The benefits in both strategies rise and go in the direction of the ideal policy as the number of episodes rises.



Conclusion

This project has applied Reinforcement Learning to the Traveling Salesman Problem with and without traffic constraint. In this project, two (Q-Learning and SARSA) of the most used algorithms have been implemented to solve the Travelling Salesman Problem. Each algorithm is simulated using python by creating our own environment which includes a set of states, actions, and rewards. Multiple scenarios have been created for 5, 50, 100 and 500 cities using each algorithm with and without traffic constraint.

References

1. Variants of travelling salesman problem: A survey. IEEE Xplore. (n.d.). Retrieved November 3, 2022, from https://ieeexplore.ieee.org/document/7033850

2. Christian, H., Hector, P., & Owais, S. (n.d.). *OR-gym: A reinforcement learning library for operations research problems*. Retrieved November 3, 2022, from https://arxiv.org/pdf/2008.06319.pdf

3. Openai. (n.d.). *Openai/gym: A toolkit for developing and comparing reinforcement learning algorithms.* GitHub. Retrieved November 3, 2022, from https://github.com/openai/gym

4. https://medium.com/betacom/travelling-salesman-problem-with-reinforcement-learning-eac425be87aa