CSE 4/546 Reinforcement Learning

Intelligent Traffic Signal Control – An Adaptive and Scalable Approach

Presented by Team 21
Anurag Hruday, Shreshta Gundoju, Gautham Krishna

Introduction

Traffic congestion is a pervasive problem in urban areas, leading to increasing travel times, economic losses as traditional traffic signals rely on predefined rules or heuristic approach. This project aims to find a solution to this using Deep reinforcement learning methods by learning from real-life traffic data, the proposed system will learn to reduce vehicle wait times.

Environment Setup

SUMO (Simulation of Urban MObility), an open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks. The suite includes a variety of tools that assist in tasks like network import, route calculations, traffic scenario visualization, and emission calculations. The XML configuration (*twoone.net.xml*) defines the road network, including the layout, traffic light logic, and connections between lanes and edges.

a) Sumo Components used to build Traffic Lights Simulator

- **Network elements (*Netedit*)**: The XML describes the traffic network including edges, lanes, junctions, and connections, which are foundational elements within SUMO.
- **sumo-gui**: The graphical user interface of SUMO is used, allowing interactive simulation and visualization of the traffic network.
- **Traffic Light Logic (*tlLogic*)**: Defined in the XML to control traffic lights at junctions.

b) Simulation Control

- *TraCI* operates on a client-server model, allowing real-time interaction with traffic simulations by connecting to SUMO (or SUMO-GUI) as a server on a specified port, enabling dynamic control over traffic elements like vehicles and traffic lights in response to the environment's state.
- It supports multiple client connections, with the ability to specify the maximum number of clients and determine their execution order through the *SetOrder* command based on assigned unique integers.
- Clients control the simulation through a series of commands, including executing simulation steps, modifying the state of various elements, and subscribing to specific simulation variables for automatic updates, improving performance.
- *TraCI* offers a rich command set with both get and set capabilities, allowing for detailed interaction and modification of virtually every component within the simulation, providing extensive control over the simulation environment.
- **Simulation Control and Commands -** Clients control the simulation through a series of commands, such as:
  1. **Simulation step execution:** Advances the simulation by one step.
  2. **Change state commands:** Modify the state of vehicles, traffic lights, etc.
  3. **Subscription services:** Clients can subscribe to specific simulation variables and get updates automatically, improving performance by reducing the need for repeated queries.

Fig A) Starting a connection to SUMO in TraCI:

- After starting SUMO, clients connect to SUMO by setting up a TCP connection to the appointed SUMO port.
- TraCI supports multiple clients, and each client should issue a SetOrder command before the first simulation step to establish the execution order.

Fig B) Closing a connection to SUMO in TraCI:

- The client is responsible for shutting down the connection using the close command.
- When all clients have issued the close command, the simulation will end, freeing all resources.

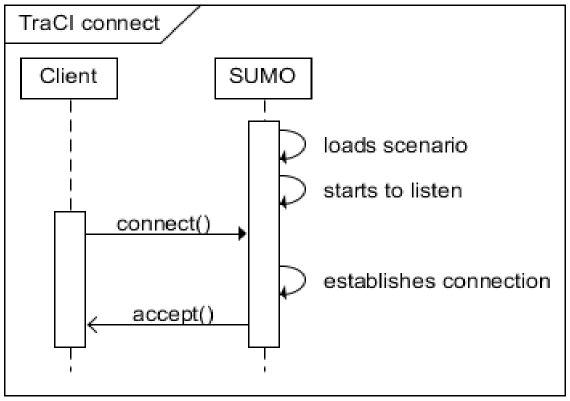


Fig A)

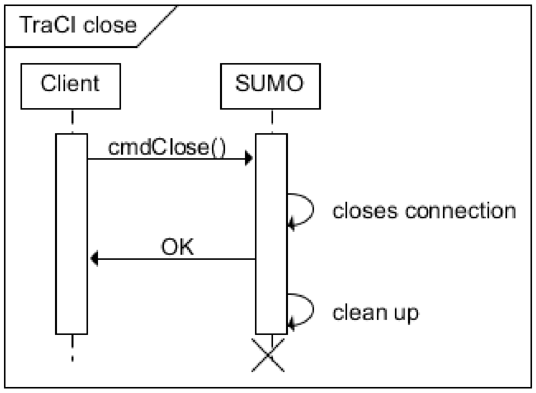
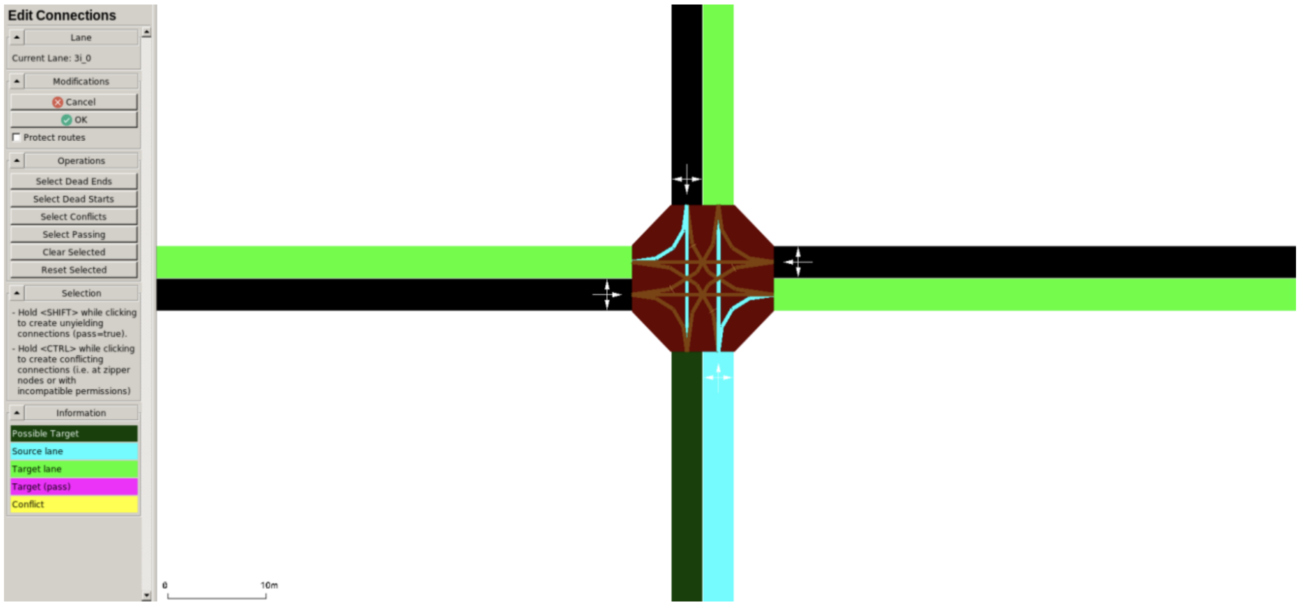


Fig B)

c) Network Configuration

- **Edges**: Represents roads in the network, each with unique ids.
- **Lanes**: Subsections of edges with attributes such as speed limit and length.
- **Junctions**: Intersections where lanes meet, some of which are controlled by traffic lights and
- others that operate on priority rules.
- **Traffic Lights**: Controlled by *tlLogic* elements with defined phases and durations.
- **Connections**: Define the permissible movements between lanes at junctions, including right-of-way rules.

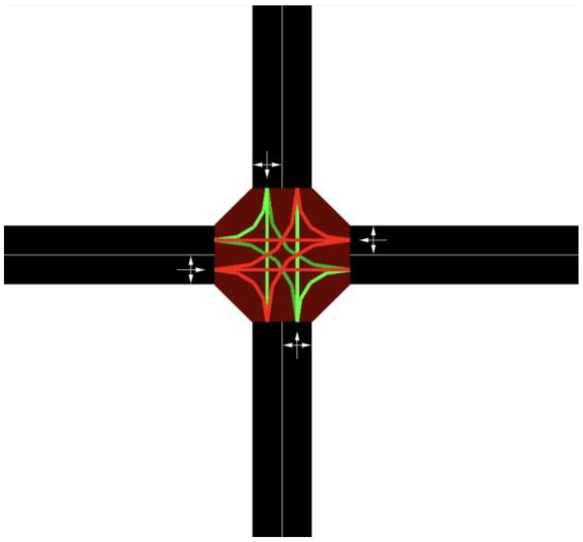
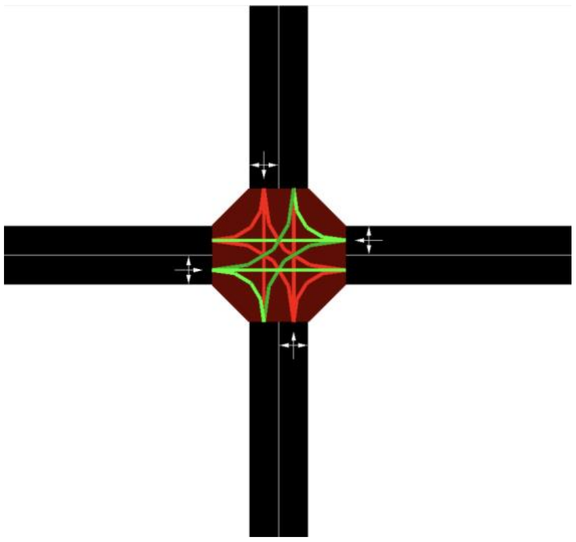


Current Lane: 3i_0

- Hold <SHIFT> while clicking to create unyielding connections (pass=true).
- Hold <CTRL> while clicking to create conflicting connections (i.e. at zipper nodes or with incompatible permissions)

d) Traffic Simulation

- **Vehicle Movement**: Vehicles are simulated within the network, with their behaviors governed by road rules and traffic lights.
- **Traffic Data Collection**: Real-time data such as the number of vehicles departed, arrived, and average speeds are collected for analysis.
- **Traffic Light Control**: The traffic lights' phases are managed to optimize traffic flow.

Figures: Traffic Signal Junction Simulation high-lighting Signal Coordination

e) Environment Impact

The dynamic environment is demonstrated in the "*generate_routefile()"* function where vehicles are generated with probabilistic tendencies, varying by vehicle types (car or truck) and routes. This randomness in vehicle generation models real-world traffic variability, impacts the learning as the agent needs to adapt to different traffic patterns each time it is trained or evaluated.

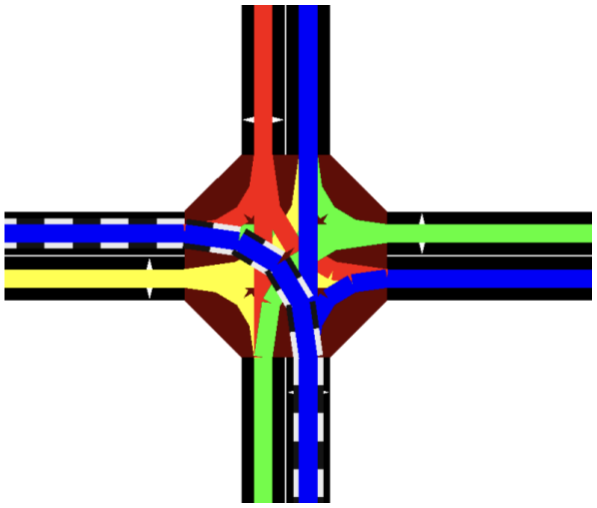


Figure: Dynamic Traffic Light Control at a Multi-Lane Intersection in SUMO Simulation

f) Action Space

The action space is defined in the "*CustomEnv*" class, where it is specified as a discrete space with two actions (0 and 1). These actions correspond to different traffic signal phases (e.g., red and green). The choice of action at each timestep impacts the traffic flow, influencing the environment's state.

self.action_space = spaces.Discrete(2) # Red and green phases only

g) Observation Space

Here in the observation space where each observation consists of 25 float32 values ranging from 0 to infinity. These values are intended to represent metrics - vehicle waiting times, densities on lanes, and current traffic light phases, which help the agent to make informed decisions.

self.observation_space = spaces.Box(low=0, high=np.inf, shape=(25,), dtype=np.float32)

h) Rewards

The reward is a combination of penalties for traffic delay and queue length, and rewards for maintaining traffic flow and safety (avoiding collisions). The agent's goal is to maximize this reward by efficiently managing traffic light phases to keep traffic smooth and minimize delays and queues.

Rewards are calculated in the "*_calculate_reward()"* function, where different aspects of the traffic scenario are taken into account as given below:

$$r^{total\_reward} = (r^{tf} - r^{td} - r^{ql} - r^{sr})$$

i. Traffic flow reward - r^{tf} :
   - Maintaining high average vehicle speeds in controlled lanes is rewarded. Higher speeds indicate smoother traffic flow with fewer stops. This motivates the agent to minimize vehicle stoppages, enhancing overall traffic efficiency.
   - Higher speeds yield a higher reward.

ii. Traffic Delay penalty - r^{td}:
   - A penalty is applied based on the number of halted vehicles. More halted vehicles indicate congestion and inefficient signal timing. This encourages the agent to reduce waiting times and improve traffic flow.
   - More halted vehicles result in a higher penalty.

iii. Queue length penalty - r^{ql} :
   - A penalty is applied for longer vehicle queues at traffic lights. Long queues can lead to extended delays and compound traffic issues. The agent learns to prevent long queues, improving traffic distribution and reducing gridlocks.
   - Longer vehicle queues result in a higher penalty.

iv. Safety reward - r^{sr}:
   - The agent is rewarded for avoiding collisions between vehicles. This prioritizes safety and reduces accident risks in traffic signal management. The reward is based on the number of prevented collisions in the simulation.

Methods

Deep reinforcement learning methods such as A2C, DQN and PPO have been used to learn traffic signal policies. All these methods are taken from stable baselines 3. It is a popular library in the reinforcement learning community that provides implementations of several state-of-the-art deep reinforcement learning algorithms.

A) A2C (Actor 2 Critic)

- Highly effective for traffic light control due to its ability to adapt dynamically to changing traffic patterns and balance traffic flow efficiently. By leveraging both the actor and the critic components, A2C can minimize traffic delays and queues, enhancing overall traffic safety and infrastructure use. Its capacity for real-time decision-making ensures optimal signal timing adjustments in response to real-time conditions.
- A2C is based on the Actor-Critic framework, where the actor determines actions based on the policy, while the critic evaluates those actions using a value function to assess their future reward potential.
- It employs an advantage function to calculate the relative quality of actions by comparing the expected reward of a specific action to the average reward for that state, helping to stabilize learning and reduce variance in updates.
- Unlike its asynchronous variant, A2C updates its policy and value functions synchronously, aggregating experiences from multiple actors before performing an update, which enhances learning stability.
- The algorithm effectively balances exploration of new strategies with exploitation of known profitable strategies, a crucial trade-off for success in diverse and dynamic environments.
- A2C is versatile, performing well in scenarios with both discrete and continuous action spaces, making it applicable to a broad range of complex tasks, including robotics and strategic games.

Stable Baselines 3 -A2C
Actor Network:

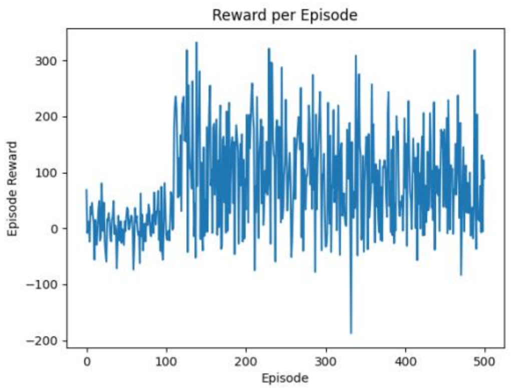
- Consists of 2 hidden layers with 64 units each by default.
- The activation function for the hidden layers is typically ReLU.
- The output layer uses the CategoricalDistribution class to compute the policy distribution over actions.
- Optionally, you can specify different architectures using the net_arch parameter, allowing customization of the number of hidden layers and units.

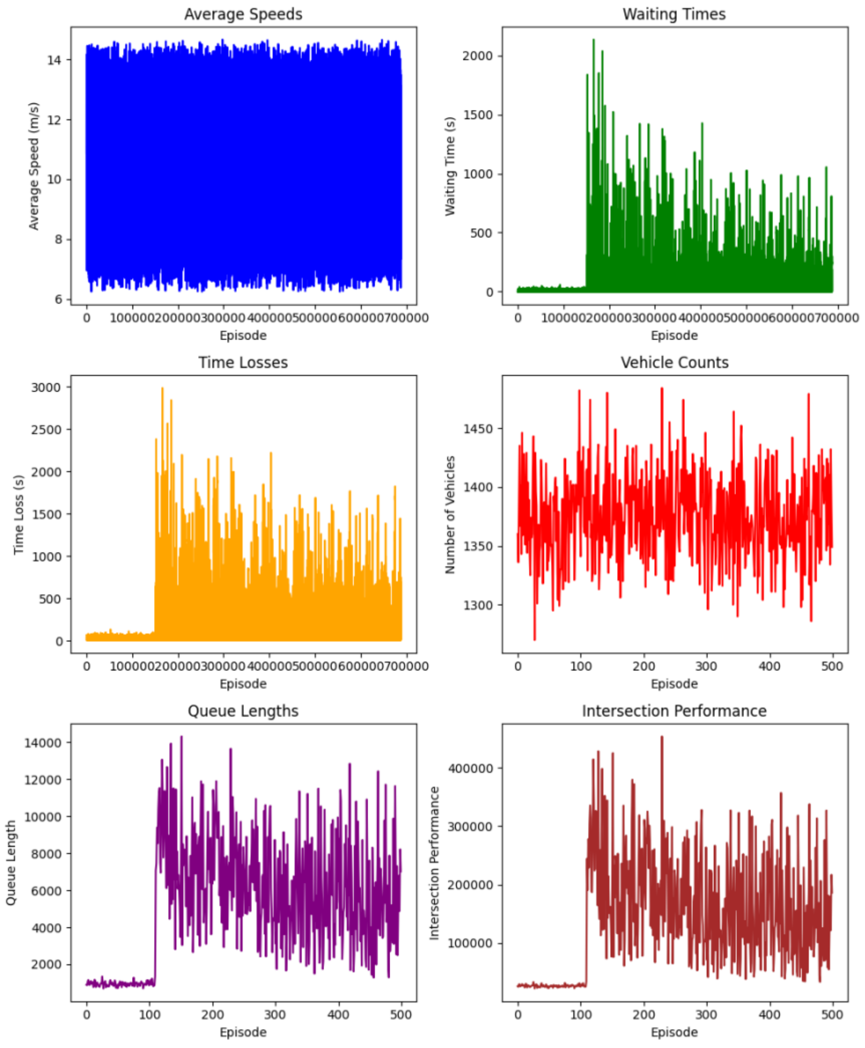
Critic Network:

- Shares the same architecture as the actor network by default.
- Consists of 2 hidden layers with 64 units each by default.
- The activation function for the hidden layers is typically ReLU.
- The output layer computes the state value.

Results using A2C:
Training

- It's important to recognize the challenges posed by a dynamic environment like SUMO, where traffic patterns and conditions can vary significantly from one episode to another.
- The "Reward per Episode" graph from the training phase displays the rewards obtained over 500 episodes. This graph illustrates significant fluctuations in reward values, indicating varying degrees of success in managing the traffic efficiently across different episodes.
- The rewards exhibit a high degree of variability, with values ranging from approximately -200 to over 300. This suggests that the complexity of the traffic scenarios since we are taking real-time traffic data which highly dynamic, the diversity in traffic patterns, or suboptimal exploration strategies.
- There appears to be an overall slight upward trend in the graph despite the noise, it could suggest that the agent is slowly learning and adapting its policy.

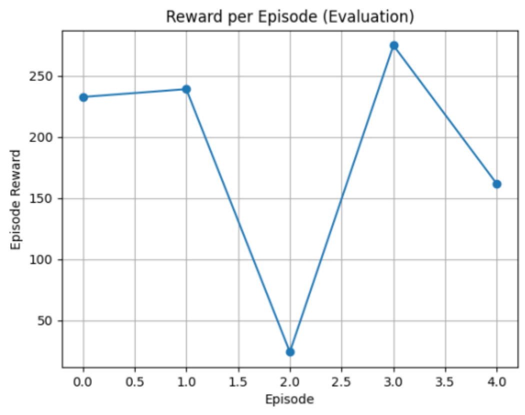




- The below series of plots for the A2C model trained in the SUMO traffic simulator provide various metrics that depict the model's impact on traffic management over episodes. The "Average Speeds" graph shows relatively stable high speeds, indicating efficient traffic flow under the agent's policies. In contrast, the "Waiting Times" and "Time Losses" graphs display a decreasing trend, suggesting improvements in reducing traffic delays as the model progresses. The "Vehicle Counts" plot exhibits variability but maintains a consistent range, which points to stable traffic volume management. Lastly, the "Queue Lengths" and "Intersection Performance" plots reveal fluctuations, which suggest that while the agent manages some intersections effectively, others may need slight policy refinement to enhance overall traffic management.

Evaluation

- The evaluation phase of your A2C model in the SUMO traffic simulator reveals notable fluctuations in performance across episodes, as seen in the "Reward per Episode (Evaluation)" graph.
- The agent generally performs well, except for a significant dip in rewards during one episode, indicating a scenario where the learned policy was less effective. This variability underscores the need for the policy to handle diverse and complex traffic conditions more consistently.
- Overall, the evaluation suggests that while the agent can manage traffic effectively in many scenarios, there is room for optimization to enhance its reliability and adaptability across a broader range of traffic conditions.



B) Proximal Policy Optimization (PPO)

- Effective in scenarios of continuous decision making and adaptability.
- Uses a clipped objective function to limit the size of the policy updates.
- It is a policy gradient method that optimizes a policy directly and is designed to provide robust learning in the environments with large action spaces by maintaining the balance between exploration (trying new actions) and exploitation (using profitable actions) which optimizes traffic flow and its efficiency in complex urban networks.
- By limiting the size of policy updates the system gradually learns and the new strategies does not disrupt traffic flow adversely.

Stable Baselines 3 – PPO

The network architecture used in the Proximal Policy Optimization (PPO) algorithm in Stable Baselines3 depends on the policy class chosen during the initialization of the PPO agent. Stable Baselines3 provides several policy classes for different types of observations, including:

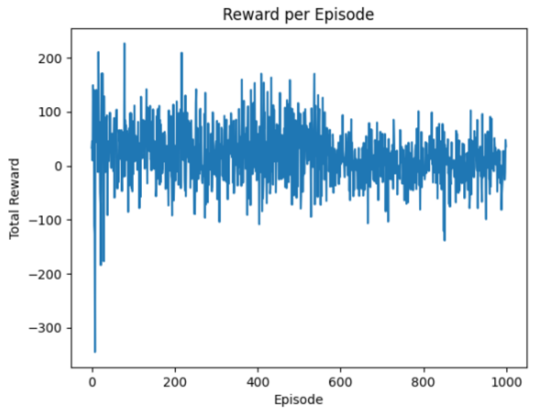
1. *MlpPolicy*: This policy class is used for environments where observations are flat vectors, such as classic control environments or environments with continuous observations.
2. *CnnPolicy*: This policy class is suitable for environments where observations are image inputs, such as Atari games or environments with pixel-based observations.

3. *MultiInputPolicy*: This policy class is used when observations are a dictionary containing multiple types of inputs, such as images and additional features.

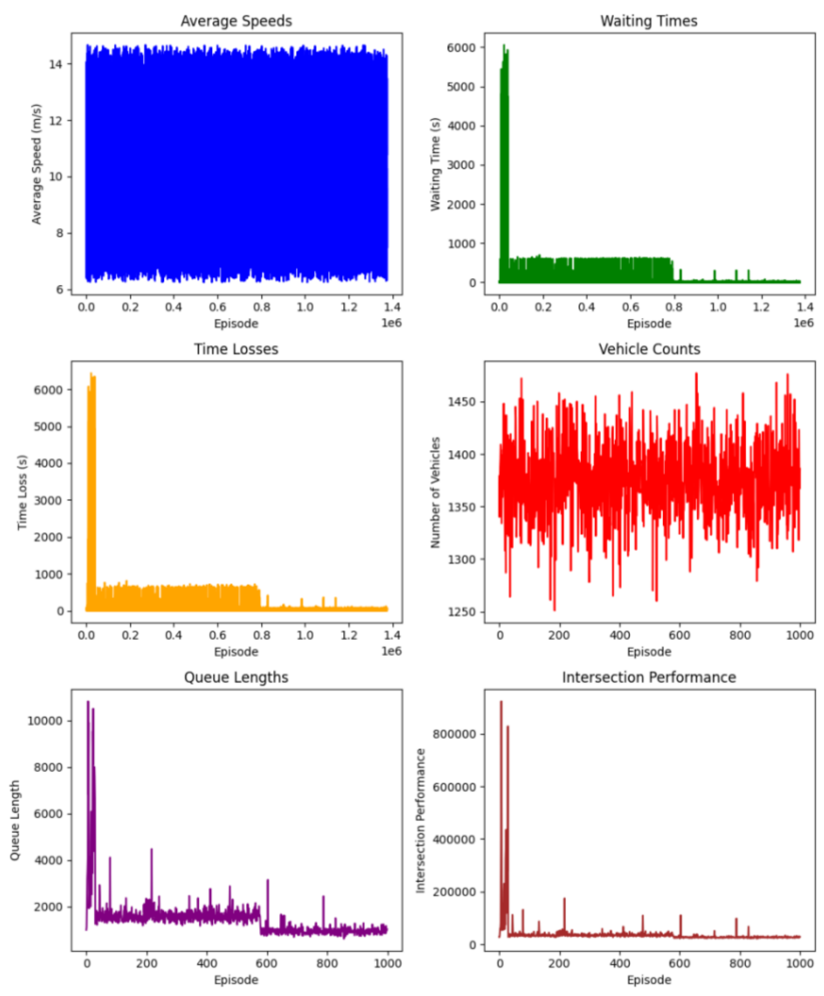
Each of these policy classes can be further customized with different network architectures specified by the *net_arch* parameter during initialization. By default, these policy classes use feedforward neural networks, but you can customize the architecture by passing a list of integers specifying the sizes of hidden layers or a dictionary specifying different architectures for the actor and critic networks.

Results using PPO:
Training

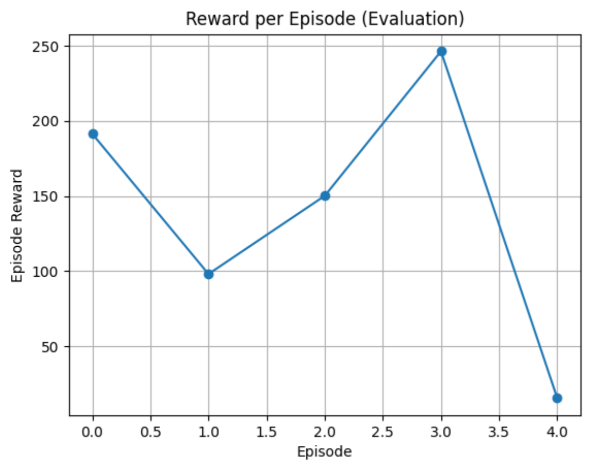
- Initial episodes show a significant reward drop as the agent explores strategies and experiences penalties.
- Rewards stabilize later, indicating a relatively stable policy, but predominantly negative, suggesting room for improvement.
- Consistent performance in later episodes implies a steady policy, potentially converged.



- The various plots from the PPO model applied to the traffic lights simulation environment offer a diverse set of insights across various traffic metrics. **Average Speeds** are consistently high, indicating the PPO model effectively maintains fluid traffic flow. In contrast, the **Waiting Times** and **Time Losses** show a sharp decline in the initial episodes, suggesting rapid learning and adaptation by the model to minimize delays. **Vehicle Counts** remain relatively stable with slight fluctuations, demonstrating the model's consistent handling of traffic volume. However, **Queue Lengths** and **Intersection Performance** indicate some variability; particularly, the sharp peaks early in the **Intersection Performance** suggest instances of suboptimal intersection management. Overall, these metrics reflect the PPO model's capability to enhance traffic efficiency.



Evaluation



- The evaluation graph for the PPO model in a traffic simulation environment shows substantial variability in rewards, indicating inconsistency in the model's performance across different scenarios. The rewards peak notably in one episode, demonstrating optimal performance, but plummet to the lowest point in the next, highlighting significant challenges. This fluctuation suggests that while the

PPO model can be effective under certain conditions, it requires further optimization to enhance reliability and consistency. Understanding the conditions leading to these peaks and troughs will be crucial for refining the model's effectiveness across varied traffic situations.

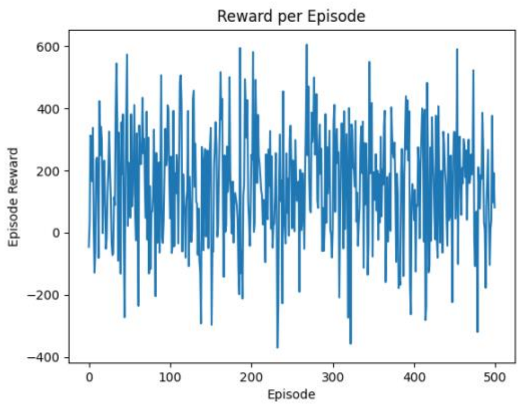
C) Deep Q Networks (DQN)

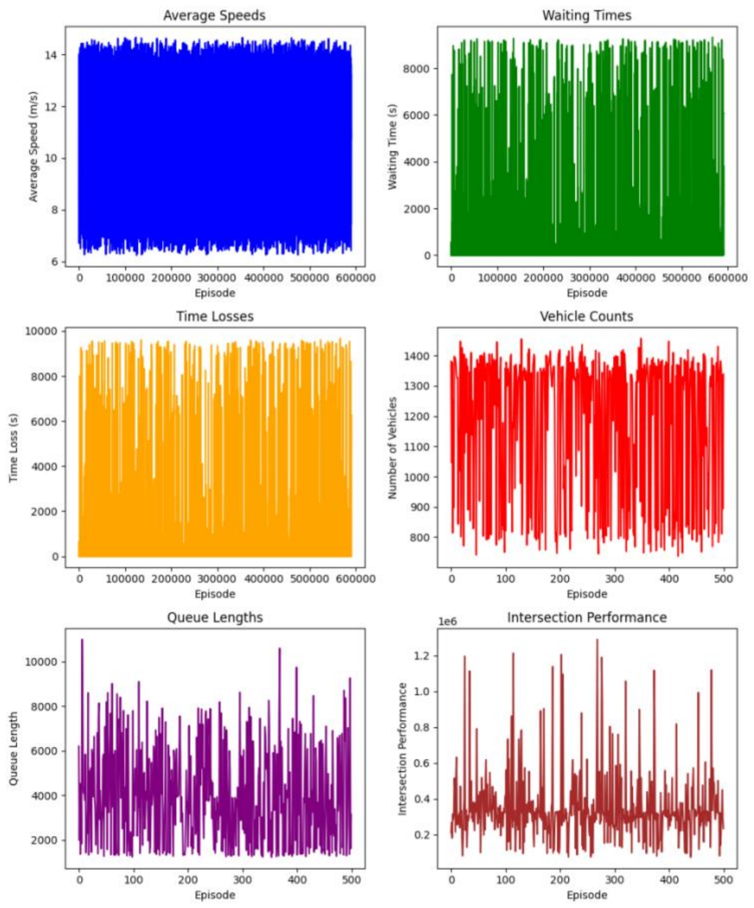
- Effective for traffic management due to its ability to handle complex, high-dimensional inputs such as vehicle counts, waiting times, and signal timings, which are crucial for optimizing traffic flow across various intersections.
- DQN employs a deep neural network to approximate the optimal action-value function, allowing it to learn complex policies for traffic management tasks.
- Through iterative updates, DQN learns to maximize cumulative rewards over time, effectively improving traffic flow and reducing congestion in urban areas.
- Beyond traffic management, DQN has been successfully applied in various domains, including robotics, gaming, and finance, showcasing its versatility and effectiveness in solving complex decision-making problems.
- Incorporates techniques like Experience replay and target networks to enhance training stability and convergence.
- Challenges include scalability to larger environments and improving sample efficiency, but future directions aim to enhance DQN's performance through techniques like distributional reinforcement learning and incorporating domain knowledge

Results using DQN:

Training

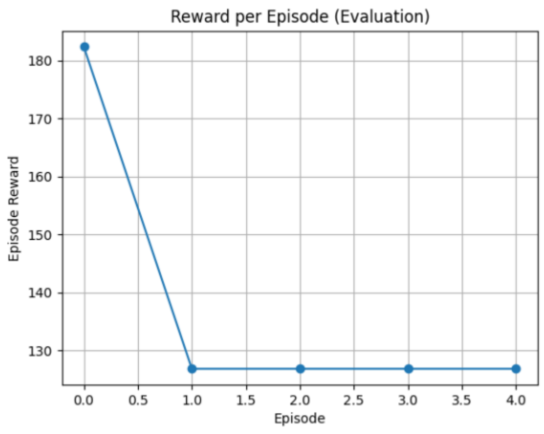
- The "Reward per Episode" graph for the DQN model in the traffic lights simulation environment illustrates significant variability in performance, with rewards fluctuating between approximately -400 and +600. This indicates that the model alternates between effective and poor control strategies across different episodes. There is no evident trend towards stabilization, suggesting that the DQN model has not yet converged to an optimal policy. The presence of both high peaks and deep troughs in the rewards highlights the potential for further refinement of the learning algorithm and reward function to enhance consistency.





- The additional performance metrics for the DQN model in the traffic lights simulation environment reveal a range of outcomes. **Average Speeds** are consistently maintained at a high level, suggesting that the model effectively promotes faster traffic flow. **Waiting Times** and **Time Losses** both display high initial values but show a trend towards stabilization, indicating gradual learning improvements over time. **Vehicle Counts** fluctuate widely, which could indicate variable traffic volumes or model responses to changing conditions. **Queue Lengths** and **Intersection Performance** demonstrate considerable variability, implying that while the model manages some aspects of traffic control well, it struggles with consistency in queue management and intersection efficiency, highlighting areas for potential enhancement.

Evaluation



- The "Reward per Episode (Evaluation)" graph for the DQN model shows a sharp decline in performance from the initial episode to subsequent ones during evaluation in the traffic simulation environment. The rewards start at a high of approximately 180 and steadily decrease to stabilize around 140, suggesting the model may be particularly effective under specific conditions but struggles to generalize across varying traffic scenarios. This indicates a need for further training or adjustments to the DQN's policy or reward structure to maintain high performance across different episodes.

Challenges

a) Performance Considerations

Using *TraCI* introduces overhead due to the communication between the client and the server, especially when frequent commands are sent or received. For high-performance needs, consider using libsumo, which embeds the simulation directly within the client application, thus eliminating network delays.

b) Troubleshooting Common Issues

- **Delayed Simulation Start:** We need to ensure all clients are connected before the simulation starts, as SUMO waits for all connections.
- **Output File Locking:** Sometimes, output files may be locked if accessed simultaneously by multiple clients or prematurely before the simulation ends. We need to ensure that all clients properly close their connections and the simulation fully ends before accessing output files.
- **Unresponsive Server:** If SUMO does not respond to *TraCI* commands, we need to frequently check if the correct port is being used and there are no network issues and also ensure that the server was started with the correct command-line options for *TraCI* control.

Conclusion

After running DQN, A2C and PPO on the traffic simulation we see that

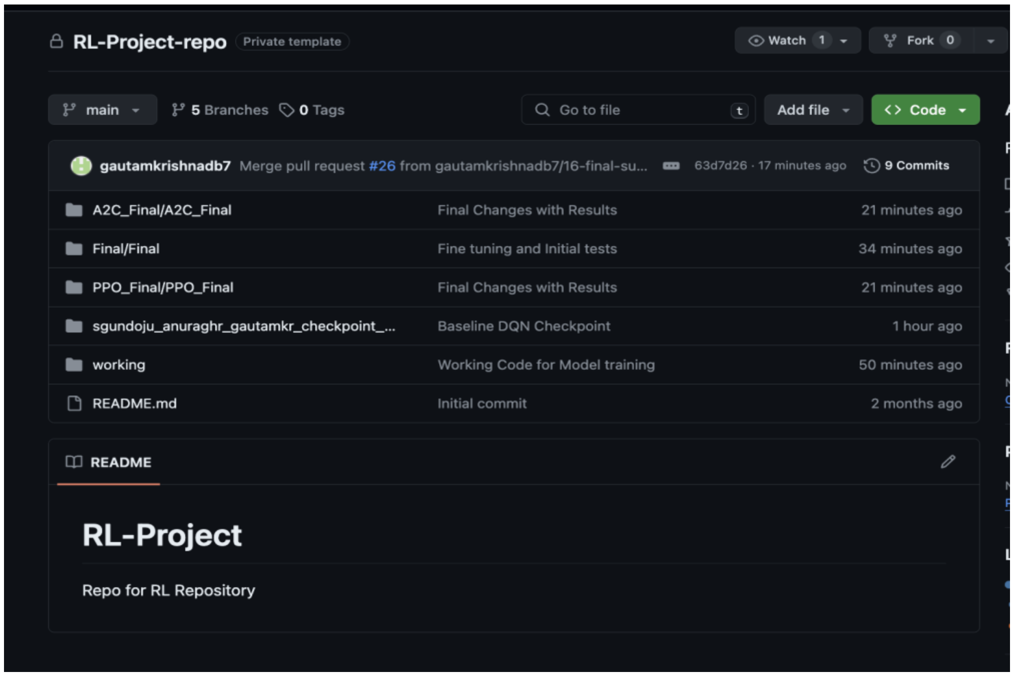
- **DQN:** Used neural network to develop the policy that reduces wait times and enhances overall traffic condition**.**

- **PPO:** Provides method to learn traffic light sequences that considers vehicle movements and scenarios at the junctions.
- **A2C:**
- A2C, DQN and PPO integration leverage their strength enhancing traffic management systems to be responsive and strategically be aware of broader traffic patterns.

Team Member Project Part Contribution

1. Anurag Hruday – 33.33%
2. Gautam Krishna - 33.33%
3. Shreshta Gundoju - 33. 33%

Github Repository



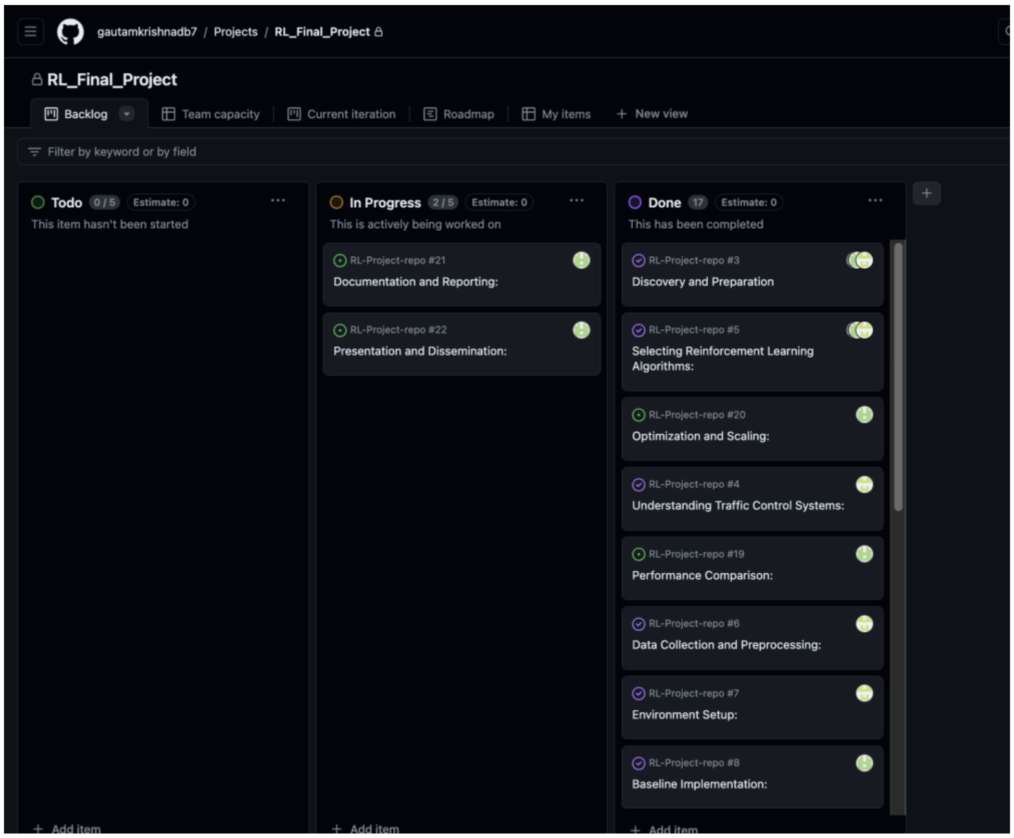
gautamkrishnadb7 Merge pull request #26 from gautamkrishnadb7/16-final-su... 63d7d26 · 17 minutes ago 9 Commits

A2C_Final/A2C_Final	Final Changes with Results	21 minutes ago
Final/Final	Fine tuning and Initial tests	34 minutes ago
PPO_Final/PPO_Final	Final Changes with Results	21 minutes ago
sgundoju_anuraghr_gautamkr_checkpoint_...	Baseline DQN Checkpoint	1 hour ago
working	Working Code for Model training	50 minutes ago
README.md	Initial commit	2 months ago

README

RL-Project

Repo for RL Repository


gautamkrishnadb7 / Projects / RL_Final_Project
Backlog
Team capacity
Current iteration
Roadmap
My items
New view
Filter by keyword or by field
Add item

RL_Final_Project

Todo 0 / 5 Estimate: 0
This item hasn't been started

In Progress 2 / 5 Estimate: 0
This is actively being worked on

RL-Project-repo #21
Documentation and Reporting:

RL-Project-repo #22
Presentation and Dissemination:

Done 17 Estimate: 0
This has been completed

RL-Project-repo #3
Discovery and Preparation

RL-Project-repo #5
Selecting Reinforcement Learning Algorithms:

RL-Project-repo #20
Optimization and Scaling:

RL-Project-repo #4
Understanding Traffic Control Systems:

RL-Project-repo #19
Performance Comparison:

RL-Project-repo #6
Data Collection and Preprocessing:

RL-Project-repo #7
Environment Setup:

RL-Project-repo #8
Baseline Implementation:

References

1. Eclipse SUMO Documentation - https://sumo.dlr.de/docs/index.html
2. Xiao Yang Liu, Ming Zhu & Anwar Walid - Deep reinforcement learning for intelligent traffic signal control (2023)
3. Yue Zhu, Mingyu Cai, Chris Schwarz, Junchao Li, Shaoping Xiao -Intelligent Traffic Light via Policy-based Deep Reinforcement Learning
4. Liben Huang, Xiaohui Qu - Improving traffic signal control operations using proximal policy optimization
5. CSE 4/546 Reinforcement Learning Slides by Alina V.
6. Stable baselines 3 documentation - https://stable-baselines3.readthedocs.io/en/master/index.html
7. SUMO GitHub - https://github.com/eclipse-sumo/sumo?tab=readme-ov-file
8. Demo Code by SUMO - https://github.com/eclipse-sumo/sumo/tree/main/tests/complex/tutorial/traci_tls