Comparing the Performance of Agents using different Observations on the Traffic Signal Control Task

Motivation

In a Model-Free Reinforcement Learning setting, there can be many different representations one can have for a state (also called observation in a multi agent case). Generally, the only limitation on choosing a state representation is that it should satisfy the Markov property. That leaves us with a lot of different state representations to try from, but are all state representations created equal? No. Depending on the function approximator chosen for representing the Q function, the set of state representations that are relatively easy for the function approximator to *comprehend* shrinks.

We are trying to find the effect of state representation on the agent performance in the traffic signal control domain. Two main objectives of the project are:

1. Establish Short-Range Temporal Scan (TDTSE) performance baseline.
2. Change representation to Queue Length Observation. Compare results with TDTSE Observation.

Background

DQN

Deep Q Learning (or DQN) is an extension of Q Learning that uses neural networks as function approximators. The neural networks learn the Q values for different `state`, `action` pairs using the transitions generated by the agent environment interaction. For finite and discrete action space, this neural network takes in current `state` of the agent as the input, and outputs the `Q` value for each action as the output.

Similar to how many tricks were required to make neural networks usable for supervised learning, many tricks are also required to make neural networks usable in the reinforcement learning setting. Some the tricks used in the DQN implementation include: Double DQN, Experience Replay, and others.

Code

Code is Confidential!!

Environment

Simulator

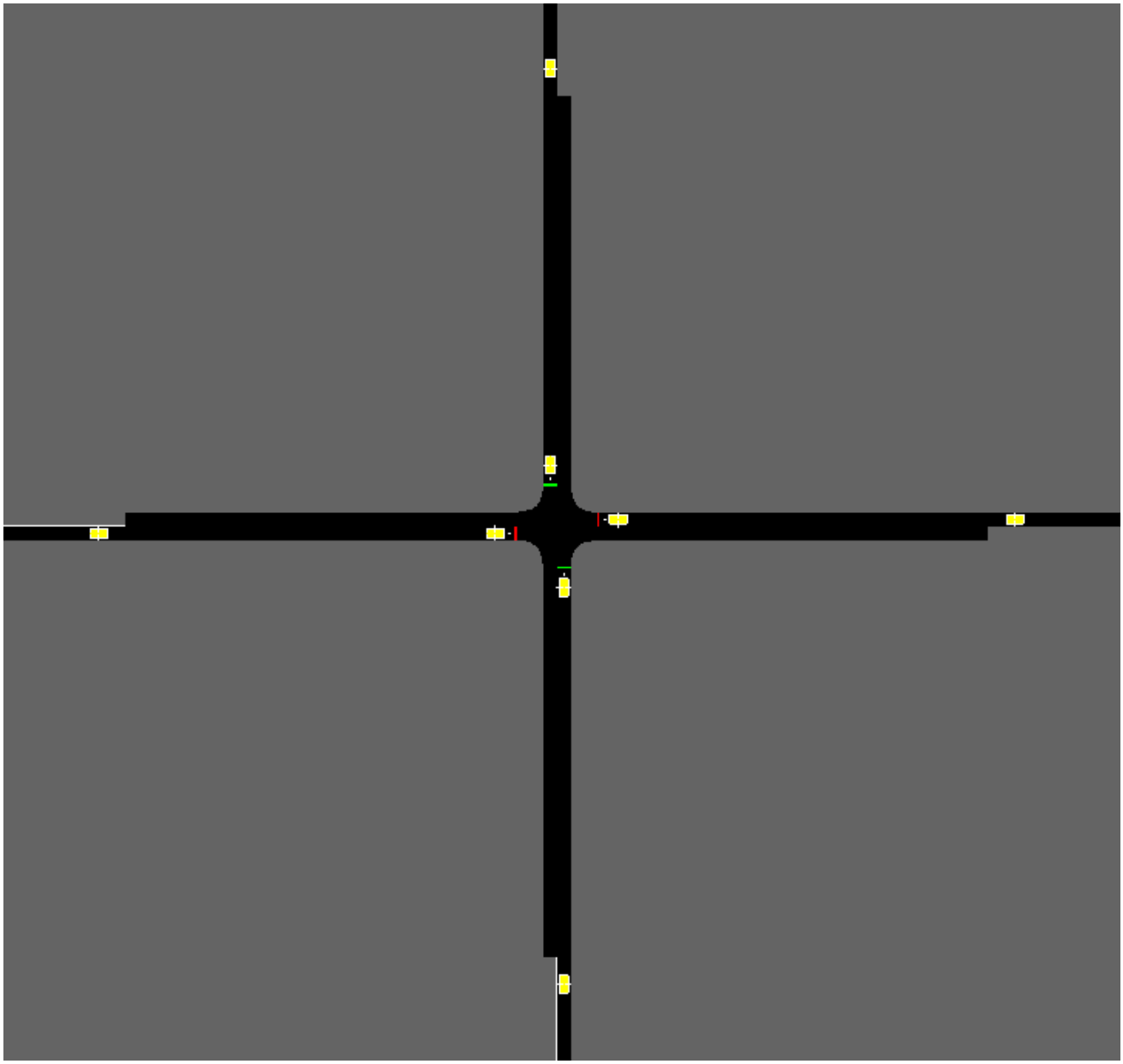
We use the SUMO traffic simulator here. https://sumo.dlr.de/docs/ (https://sumo.dlr.de/docs/)

Sumo is an open-source package that can perform microscopic and continuous traffic simulations. We use flow to extract out the Sumo APIs, but scenarios such as dealing with traffic loop detectors needed some extra tinkering.

Network

Network is single intersection. In Reinforcement Learning, exploration generally begins with the simplest of experiments. This is because RL is very hard to train, and performing a check on the simplest of scenarios is a good sanity check. If needed even this simple scenario can be scaled to more complex tasks with minor changes. This includes changing number of lanes, changing the traffic flow parameters, and other network properties.

Single intersection with `8` induction based loop detectors, two detectors on each lane at a distance of `5` and `100` meters from the intersection.

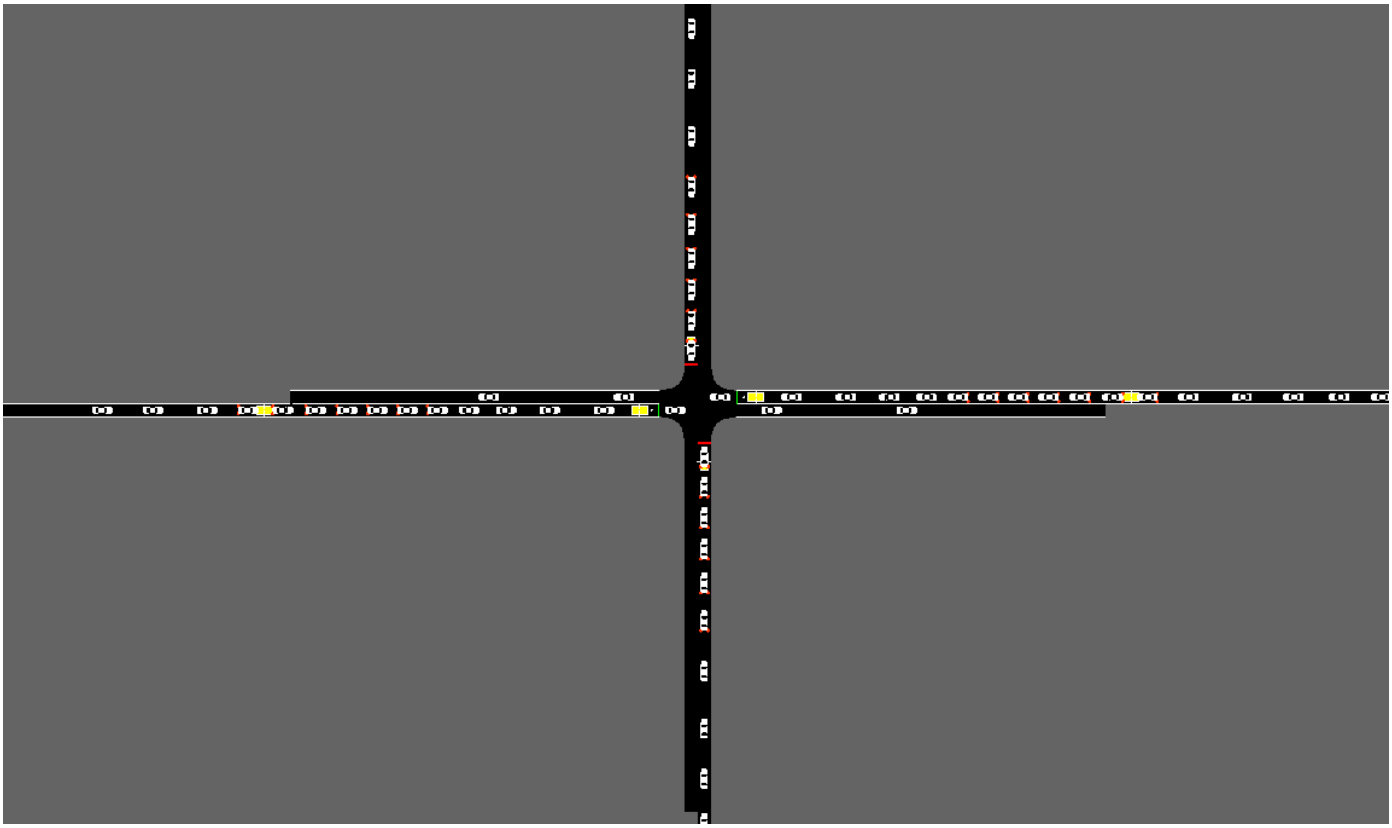


Traffic Inflow

To perform the simulation we need to provide the amount of vehicles that enter the simulation. Earlier traffic inflow was defined in the terms of `vehsPerHour` parameter given by SUMO. This introduces vehicles into the simulation in an equally spaced manner. This resulted in the same kind of traffic entering the lanes through the East and West directions (same for North and South).

To ensure this does not happen, we use `probability` parameter for introducing inflow into the network. We used the value of `probability=0.3` as this results in reasonable amount of congestion into the network. An image of the saturated network is shown below. Since free-flowing uncongested networks are not that interesting use-cases, we run our DQN trainings on this congested network. This value of probability ensures that there may be certain moments where the network becomes free, but it is congested most of the time. This scenario is also a good stress test for the controller.

SUMO Simulation snapshot when `probability=0.3` . `probability` is probability of spawning new vehicles at a lane.



Action Space

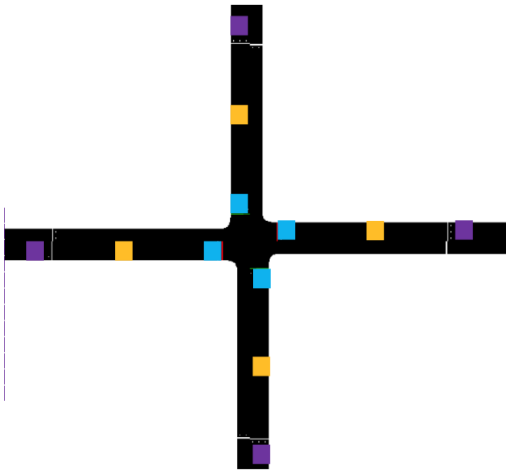
Action for the agent is `Discrete(2)`. The agent can perform two actions which are `Extend` and `Change`. `Extend` extends the current phase to the next timestep, while `Change` cycles the phase in the next timestep to the next phase in the phase list.

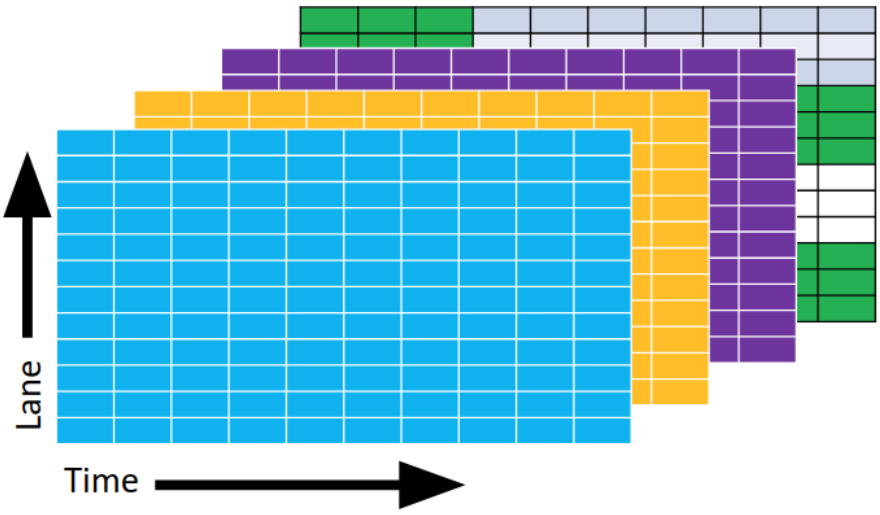
Observation Space

We have two observation spaces we would like to compare.

1. **TDTSE Observation:**

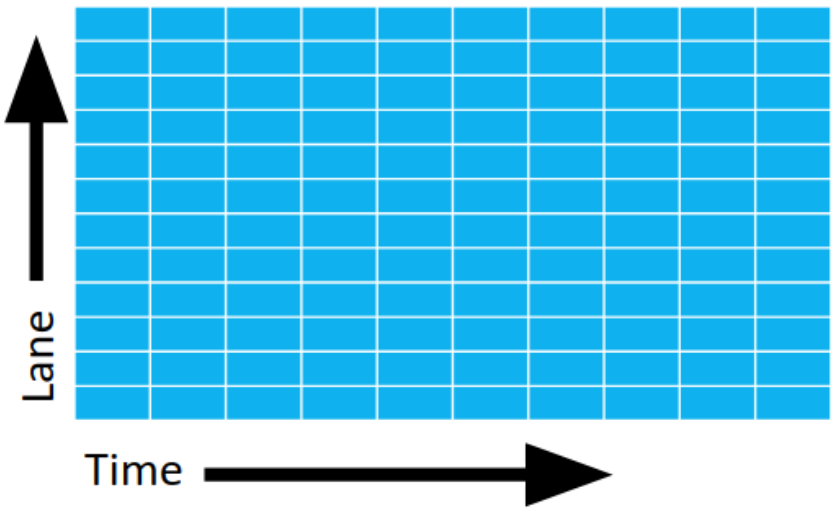
The TDTSE observation fills in 1's and 0's within the matrix shown below based on the occupancy on vehicles on the lane detectors shown in the network layout.





2. **Queue Length Observation:**

The Queue Length Observation fills in the Queue Length of each lane in the matrix below and the X-axis which represents time ensures that the representation also encodes past information if need by the controller.



Reward Function:

1. **Queue Length metric**: We use the Queue Length metric for our reward function. Queue Length metric is the number of vehicles that are in the queue in all the incoming lanes of the intersection. Vehicles that are below `0.3m/s` are considered part of the queue. Since we would like to minimize Queue Length, it's negative value is taken as the reward.

2. **Travel Time metric**: Represents the time taken by each vehicle to reach its intended destination. Contrary to Queue Length Metric which is evaluated at each timestep of the simulation, Travel Time metric is evaluated at the end of the episode as the sum of the travel times of all the vehicles becomes available then. This is not used as the reward function for training the DQN controllers. Since we would like to minimize Travel Time, it's negative value is taken as the reward.

We further compare both metrics in the Experiments Section.

Baselines

To establish baselines for Traffic Signal Control Task, we use a Random Baseline Controller and multiple Static-

Time Baseline Controllers. The Random Controller chooses actions randomly, therefore it is used as one of the baselines to establish the lower limit of the controller performance. But we would see later in the Experiments section that in some situations it doesn't perform as bad as one would expect, and the reasons behind it.

Contrary to Random Controller, Static-Time Controller establishes a much more useful baseline to judge our controller's performance as this controller is similar to how actual traffic signals work. In a Static-Time Controller, each phase is given a fixed time, and once the fixed time interval has passed, controller switches to next phase which also has its own fixed time interval.

When training on traffic networks, there many constants that represent inherent attributes for the network: one such constant is the `traffic_light_params` which represents the minimum and maximum duration that each phase can take in a single cycle of phases. Note this object will be different for different intersections, and can vary heavily for each phase (example below shows same for each phase), as traffic in certain directions may be higher. Also these parameters are liable to change as the flow of traffic changes.

```
traffic_light_params = {
    # phase: {'min': min_duration, 'max': max_duration}
    'rGrG': {'min': 10, 'max': 60},
    'GrGr': {'min': 10, 'max': 60},
    'yellow_duration': 3,
    'interphase_duration': 2
}
```

We assume that the provided `traffic_light_params` object is applicable in wide distributions of traffic flow including the flow at which we are training our models. We use this values of this object to create 3 baseline controllers. The three baselines are:

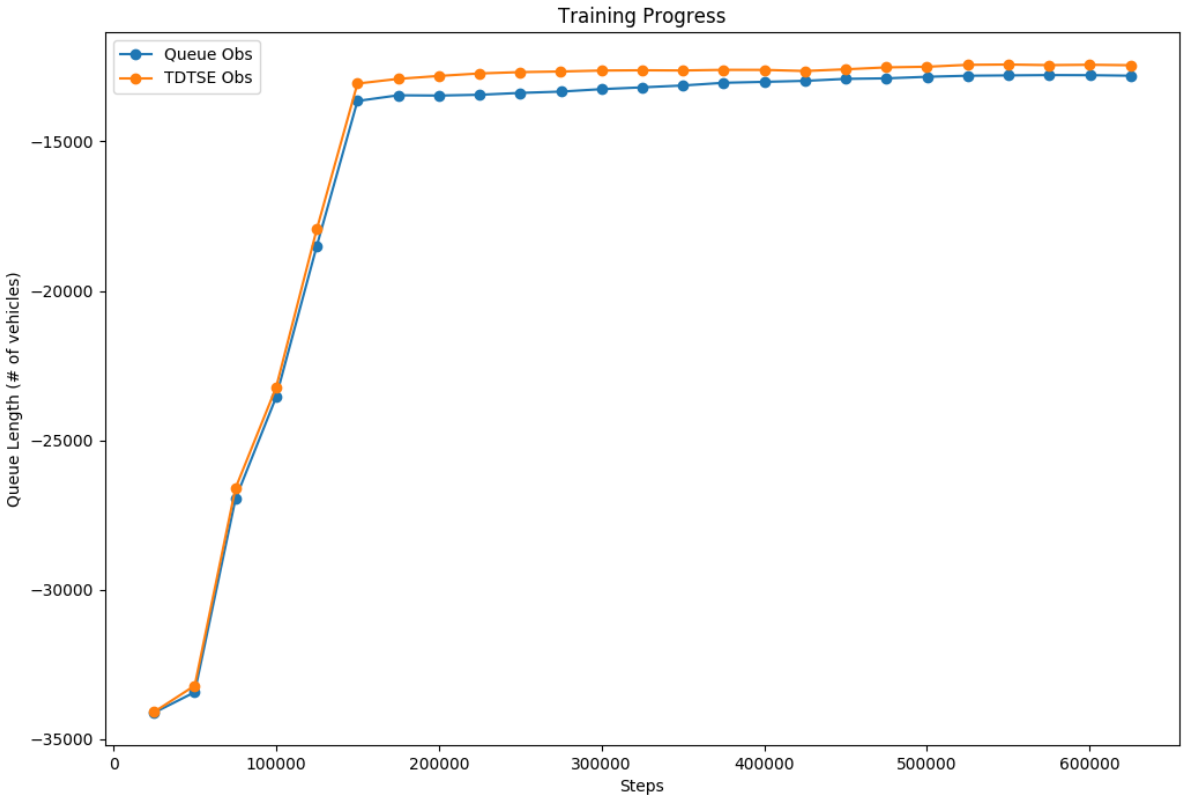
- Static-Min Baseline Controller: this controller switches the phases when minimum time of the current phase has been reached. When used with the above object, it would change phases every 10 seconds.
- Static-Max Baseline Controller: as the name implies it would switch phases when the maximum time of the current phase has been reached. With the above object it would switch phases every 60 seconds.
- Static-Mid Baseline Controller: this controller was added as a an extra baseline which switches phases at the mean of minimum and maximum time you can stay in a phase. For above case it would switch after every 35 seconds.

These four baselines (one of Random and three of Static) ensure that baselines cover a wide variety of traffic conditions.

Experiments

We train our Q network architecture over 25 iterations, with each iteration spanning 25000 timesteps of the environment. The training progress is shown in the below figure. As seen from the plot that the training starts to plateau from iteration number 6.

Figure 1: Training Progress.



Architecture

Architecture of the network used for the Q Network:

Conv2D Layer: 32 filters, kernel: [1, 8], ELU activation
Conv2D Layer: 32 filters, kernel: [1, 4], ELU activation
Conv2D Layer: 32 filters, kernel: [1, 2], ELU activation
FC Layer: Units: 128
FC Layer: Units: 2 (number of actions)

Videos

We recorded some of the videos of how the different agents were performing, and you can view the results below.

Queue Observation DQN
Video: https://drive.google.com/file/d/1jWL88Yv_mGJ_N9mP0KWeOr2t13o_vkZZ/view?usp=sharing (https://drive.google.com/file/d/1jWL88Yv_mGJ_N9mP0KWeOr2t13o_vkZZ/view?usp=sharing)

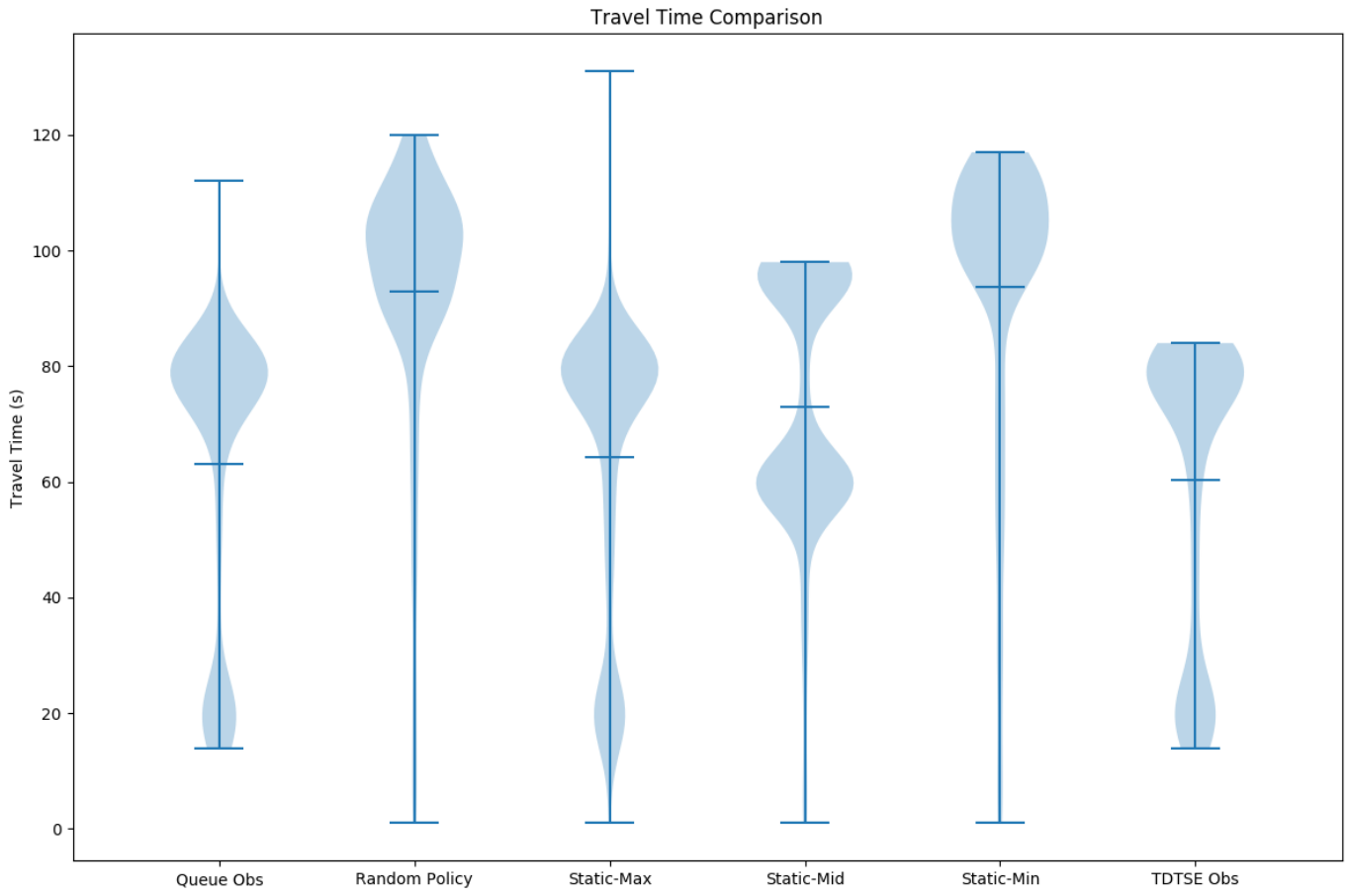
TDTSE Observation DQN
Video: https://drive.google.com/file/d/1o6f0lur739NZKwdAZ3jMJBDdIa_aEBHp/view?usp=sharing (https://drive.google.com/file/d/1o6f0lur739NZKwdAZ3jMJBDdIa_aEBHp/view?usp=sharing)

Static-Max
Video: https://drive.google.com/file/d/1zZjy6BLYQFr6R0p3tTOvmreAhQPD57UN/view?usp=sharing (https://drive.google.com/file/d/1zZjy6BLYQFr6R0p3tTOvmreAhQPD57UN/view?usp=sharing)

These are discussed alongside the graphed results below.

Graphs

Figure 2: Travel Time Comparison between different algorithms. Here we create a violin plot based on the time taken by each vehicle to reach its destination. Each violin plot has a dash in the middle that represents the mean travel time over all vehicles. We can see that TDTSE has the lowest mean value here, so TDTSE is the best controller based on the Travel Time metric. The worst controller is Random baseline controller which performs very similar to the Static-Min baseline controller.



Queue Length Observation and TDTSE Observation

Our expectation was that Queue Length Observation would perform better than TDTSE Observation, but TDTSE still performed slightly better. One reason for this is we haven't performed a through search over different network architectures that take in Queue Length Observation as input. Since the structure of this observation is different from the TDTSE observation, it maybe possible that different architectures work better than the Convoluation architectures.

Another reason for this is the inadequacies of the current Queue Length Observation implementation. Current implementation considers any vehicles that is below the speed of `0.3m/s` to be part of the queue of that lane. But if there are 5 vehicles travelling on that lane at a considerable speed, the Queue Length Observation shows the queue length as zero. This is same for the case when there are no vehicles on that lane. So current Queue Length Observation cannot differentiate between these two widely different situations.

Static-Max Controller

Here we see that the Static-Max controller is quite close in performace to the reinforcement learning algorithms. But we claim that this would not be the case if there was some differential in the amount of traffic travelling in the North-South direction with respect to the traffic travelling in the East-West direction. The nature of the Static controller is such that it cannot optimize with respect to the such conditions.

But since we are working in a high traffic flow setting, a higher performance from Static-Max controller does tell us that in these conditions it is preferable to change phases less often. Or does it?

Viewing the video for Static-Max case, one can see that when the phases are not changed for too long, queue builds on in one direction to large values. This ensures that no further vehicle may be added to this queue by the simulator, and both Travel Time and Queue length metric are kept low artificially due to this. But there is another reason why Static-Max does well in the above experiment, which will be discussed later. But this observation makes us unsure whether the Static-Max controller can still perform well if the lanes were sufficiently long.

RL Controllers

Next natural question to ask would be if the RL controllers (Queue Obs and TDTSE Obs) are also gaming the system similar to what Static-Max controller is doing. Short answer is no. As seen in the videos, these controllers change the phase after 10-15 timesteps (or seconds). So their mechanism for achieving good performance on the Travel Time metric is different from Static-Max controller.

Comparing Static Controllers

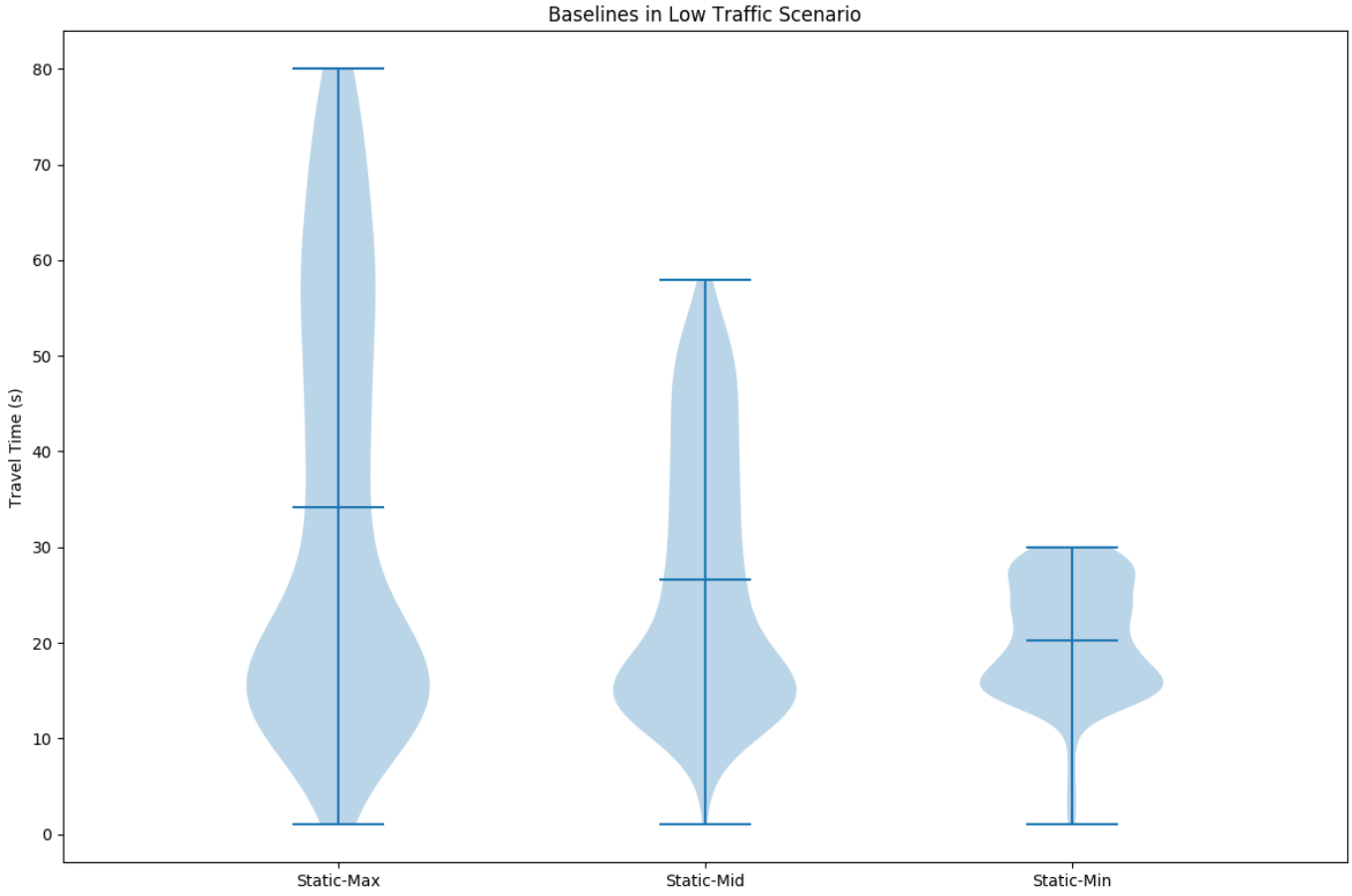
Why do we see such progression of rising performance across Static-Min, Static-Mid and Static-Max baseline controllers?

Well one reason is similar to discussed above for the Static-Max case, that is high queue lengths decrease the vehicle incoming rate artificially. But there is another reason for this jump in performance. When the network is saturated with traffic, rapidly changing between phases will increase the time taken by the vehicles to reach their destination. This is because they would have to repeatly accelerate and deaccelerate due to changing phases before they finally cross the intersection. This is why Static-Min does poorly in our heavy traffic scenario, while Static-Mid does better than Static-Min.

So by the above logic, if the traffic was extremely light the above trend should be reversed. And this is exactly what is observed (Figure 3, below) when we ran the above three Static controllers on our network with `probability` of spawning vehicles being `0.05`. This low probability replicates the low traffic scenario in our network. And here we see the opposite trend in the Static controllers. In this case the queue lengths are small, so it makes more sense to switch between phases so that all vehicles can quickly reach their destinations.

By this analysis we have established that the time given to each phase is dependent on the flow of traffic that exists in the network. This is an important conclusion and will be used in further sections.

Figure 3: Travel Time Comparison between different algorithms in a Low Traffic Scenario.



Comparing RL Controllers to Static Controllers

We mentioned previously than RL controllers exhibit different behaviour than Static-Max controller because they change their phases every 10-15 timesteps (or seconds), as compared to Static-Max which changes phase every 60 seconds. But if above is true, why is the performance of Static-Min controller (that changes phase every 10 seconds) is so different from the RL controllers. One possible reason for this is that the RL controllers exhibit different behaviors during different times of the episode. It could be possible that in the video they switch phases every 10-15 timesteps, but in other situations this duration is much larger. This would explain the overall difference in travel time metric between RL controllers and Static-Min controllers.

Phase Duration

A useful criterion to compare these controllers is the Phase Duration (PD) or the time they spent on each phase. Since in our scenario the North-South and East-West bound traffic flows are similar, when we mention PD it corresponds to the average value of PD over both the phases.

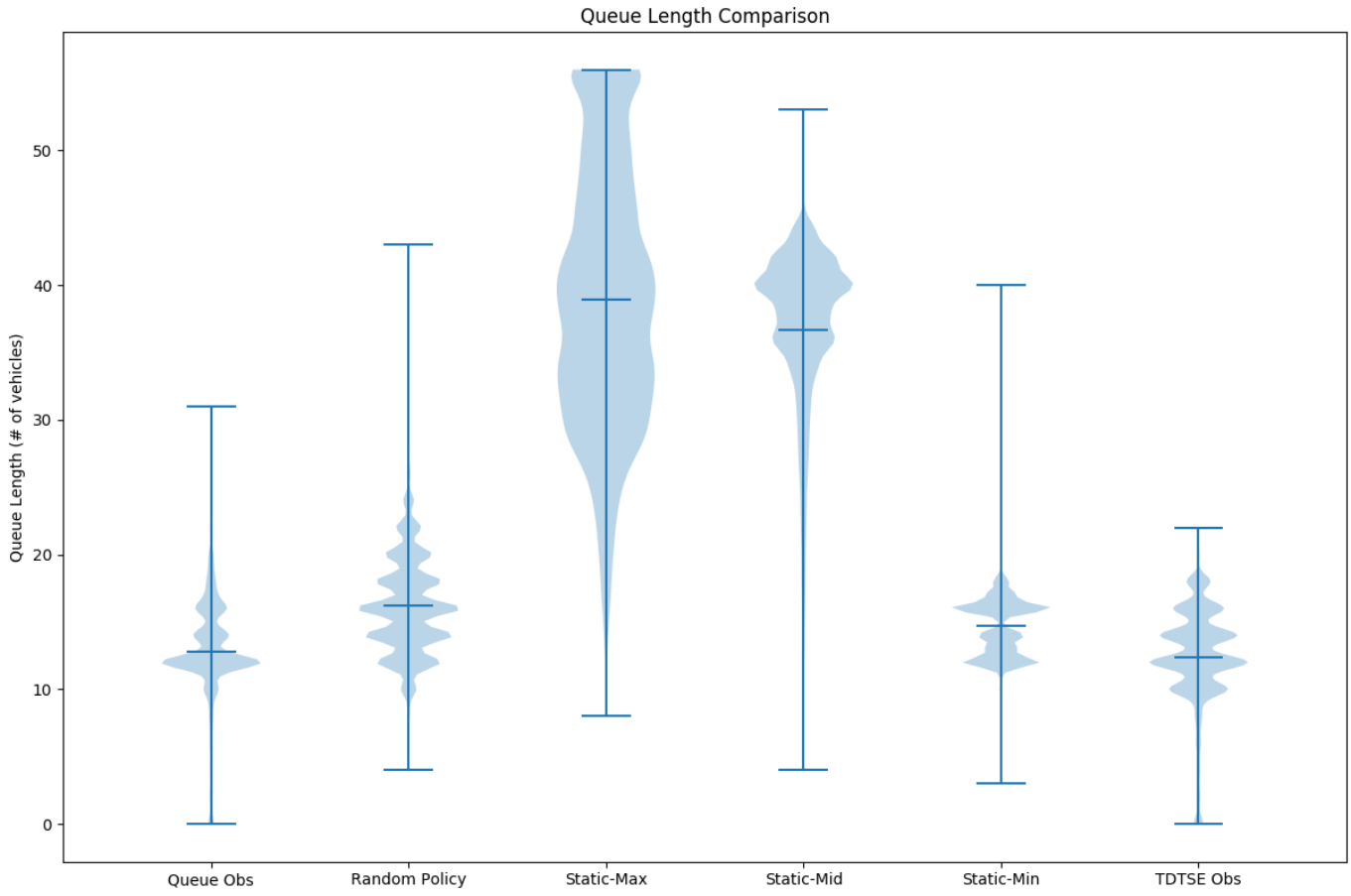
Based on this criterion, the baselines differentiate well: Static-Min has a much lower PD, while Static-Max has a large PD. And we already established in previous sections that certain PD values are optimal for certain situations, for eg. higher PD better in the high traffic scenarios. For the RL controllers, we assume that the Phase Duration changes with the conditions the controller is faced with.

A couple of interesting questions to consider with respect to Phase Duration:

- In high traffic scenarios, does it make sense to keep on increasing the Phase duration, or is there a limit above which it starts to become detrimental.
- If the RL controller's Phase Duration varies with respect to the traffic flow conditions, how exactly does it vary with the traffic flow.

Graph that would summarize this experiment:
RL controllers show different values of Phase Duration during their their life inside an episode. A violin plot of these different Phase durations can expressively represent this variation. The mean from the violin plot would also show which baseline controller is the RL controller similar to on average. Is the average phase duration closer to Static-Min controller or Static-Max controller. Unfortunately due to lack of time, this has been added to future work.

Figure 4: Queue Length Comparison between different algorithms. Lower is better. Here we create a violin plot based on the queue length that exists at each time step during the evaluation run of these controllers. Each violin plot has a dash in the middle that represents the mean queue length over all timesteps. Like in the travel time metric, TDTSE Observation Controller performs best here closely followed by the Queue Observation Controller. Worst performing controller here is the Static-Max controller.



Random Policy and Static-Min Controller

Why are their results for these two controllers so similar in both travel time and queue length metrics?

Random Policy controller randomly choosing between the provided actions which are: `Extend` and `Change`. `Extend` action extends the current phase, whereas `Change` action changes to the next phase. Since this controller selects between these 2 actions randomly, there is a high chance that the `Change` action will be choosen within the first few timesteps when it is allowed to change phases. This is very similar to Static-Min controller which changes phase at 10 timesteps. This is why these controllers show similar behaviour and metrics.

Comparing Static Controllers

We saw previously that the Static controllers varies in performance over the Travel Time metric based on the flow and density of traffic in the network. Static-Min controller does best in Travel Time metric when there is low traffic, whereas Static-Max controller does best when there is heavy traffic.
So is there a similar phenomenon when these Static controllers are compared over the Queue Length metric in the low traffic setting?

This is not the case for Queue length metric. Regardless of the traffic conditions, Static-Min always works well to reduce the overall queue length of the intersection. This may change when we have dissimilar flows across the two lanes, but for our scenario this always holds true. So Static-Min always does well on Queue Length as compared to other Static controllers.

War of Reward Functions

Which is a better metric: Travel Time vs Queue Length?

- Even though Rapid switching between phases in traffic heavy network leads to lower Queue lengths, it increases the overall Travel Time of the vehicles in the network (as seen by Static-Min Controller's performance over both the metrics). This is essentially a question of who do optimize with respect to: do we optimize with respect to the intersection or the vehicles. Queue lengths optimizes with respect to the intersection, whereas Travel Time optimizes with respect to the vehicles inside the network. Controller optimized over Queue length metric will ensure that vehicles do not stay in the intersection for long. But the controller optimized on Travel Time metric will ensure that all vehicles cross the intersection in a timely fashion. The above two objectives could be same or different, it depends on amount of vehicles in the intersection.
- In the Static-Max Baseline controller, we saw that some lanes got blocked which lead to lower incoming vehicles within the network. Queue Length as a metric is dependent on the number of vehicles in the network. While the number of vehicles in the network affects the Travel Time metric as well, it does so indirectly by causing congestion within the network.

Based on above mentioned reasons, Travel Time seems a better metric to use for judging different controllers.

But if Travel Time is a much better metric then why not use it for training the reinforcement learning controllers?

We use Queue Length metric as a reward function for training our reinforcement learning controllers because Travel Time metric is only accessible at the end of the episode. If the episode is upto 1000 steps, this is the difference between getting 1 reward value and 1000 reward values. Sparsity of rewards is one of the issues that makes Reinforcement learning algorithms difficult to converge. One possible alternative to the above issue could be first training the model over the Queue length reward, and then later fine-tuning it over the Travel Time reward. This may work because in many situations similar actions will lead to similar rewards in these metrics.

Conclusions

We aimed to compare between the different state representations that could be provided to the Q network. Our expectation was that Queue Length observation would perform better than TDTSE observation. But due to inadequacies of current Queue Length Observation it falled slightly short of TDTSE Observation.

Rather than using the Queue Length representation, we propose that we should use a Lane Occupancy Observation which consists of two parts: first is count of vehicles on that lane, and second is the average speed of the vehicles on that lane. By using this representation we will overcome shortcomings of Queue Length

Observation, and will also have more information than the TDTSE Observation which does not consider vehicle speeds. Lane Occupancy Observation will be able to differentiate between five moving vehicles on the lane, or no vehicles on the lane the scenario in which Queue Length Observation felt short in.

We saw that Travel Time metric is more useful metric than Queue Length metric from comparing performance between different Traffic Signal Controllers. We also saw that to ensure converge of training as well as faster training of DQN controllers, we still use Queue Length reward functions. It was interesting to view the difference between these metrics as they try to optimize the agent over different objectives depending upon the flow of traffic within the network.

Future Work

Due to lack of time, we could not complete the analysis we wanted to accomplish. In this section, we compile a comprehensive list of next tasks we will be working on to further complement this analysis. Working on this project has been much more insightful that was thought previously, especially the part of comparing our controller's performance to the baselines. It sprouted the idea that our controller is nothing more than a combination of different baselines such that different baselines are used in different situations by the agent. Our future work will dig deeper into this, by trying to answer why is that some baselines are preferable in some situations but not in other situations.

Next Tasks:

1. Understanding dynamics using experiments:
    - Consider to sufficiently long lanes spanning in the NS and EW directions. For fixed `probability` of spawning new vehicles, what should be the Phase Duration (PD) such that we recieve the lowest value on the Travel Time metric.
    - How would the Phase Duration (PD) change as the spawning `probability` of vehicles is varied.
    - Do the domain specific controllers (Webster, Max Pressure, others) agree with these findings.
    - Are the model-free RL controllers able to identify these optimal Phase Durations for varying level of traffic inflow.
    - Are the model-based RL controllers able to identify these nuances better than model-free ones.

2. Implement the graph that explains this experiment:
    - Violin plot of different Phase Durations that a controller exibits during its activity in an episode. Compare with other controllers.

3. Implement the Lane Occupancy Observation discussed in the Experiments section.
    - Estimate this representation by learning a regressor from the loop detector data.

4. Establish a benchmark to test different representations. Certain representations might perform better in certain situations. To create a benchmark: vary number of lanes (incoming and outgoing), vary the traffic (`probability` of spawning), and limits of `traffic_light_params` .