Assessment of Multi-Modal Reward Functions in Reinforcement Learning for Urban Traffic Control under Real-World limitations

Alvaro Cabrejas Egea
MathSys Centre for Doctoral Training,
University of Warwick & Vivacity Labs, London, UK
Email: a.cabrejas-egea@warwick.ac.uk

Colm Connaughton
Warwick Mathematics Institute
University of Warwick
Email: c.p.connaughton@warwick.ac.uk

***Abstract*— The abstract goes here.**

I. Introduction

II. Related Work

As indicated in [1], pedestrian implementation has a high impact on learning performance, being often discarded as unimportant or left for future work.

In this paper we attempt to cover this gap in the literature, providing with a performance assessment of RL agents serving both vehicles and pedestrians, using a variety of rewards in a calibrated model of a real-world junction.

III. Problem Definition

A. Markov Decision Processes and Reinforcement Learning

The problem is framed as a Markov Decision Process (MDP), satisfying the Markov property: given a current state s_t, the next state s_{t+1} is independent of the succession of previous states $\{s_{t-1}, s_{t-2}, ..., s_0\}$. An MDP is defined by the 5-element tuple:

1) The set of possible states $\mathcal{S}, s_i \in \mathcal{S}$.
2) The set of possible actions $\mathcal{A}, a_i \in \mathcal{A}$.
3) The probabilistic transition function between states $\mathcal{T}$.
4) The discount factor $\gamma \in [0, 1]$
5) The scalar Reward Function $\mathcal{R}$.

The objective of an MDP agent is to find an optimal policy π^*, mapping states to actions, that maximises the sum of expected discounted reward.

$$R_t = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \tag{1}$$

In this case, $\mathcal{T}$ is unknown, only allowing for model-free Reinforcement Learning (RL).

Model-Free RL is an area of Machine Learning covering how independent agents can take sequential decisions in an unknown environment and learn from these in order to obtain π^*. There are two main approaches: Policy-Based RL, in which states are mapped to a distribution of potential actions, and Value-Based RL, which is used in this paper and estimates the value (expected return) of the different state-action pairs under a given policy π as defined in Eq. 2.

$$V^{\pi}(s) = \mathbb{E}[R_t | s, \pi] \tag{2}$$

B. Q Learning and Value-Based RL

Q-Learning [3] is an off-policy value-based RL algorithm. For any finite MDP, it can find an optimal policy (maximise expected total discounted reward) starting from the current state [4]. Q-Learning aims to learn an optimal action-value function $Q^*(s, a)$, defined as the total return after being in state s, taking action a and then following policy π^*.

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\big[R_t | s = s_t, a = a_t, \pi^*\big] \tag{3}$$

Traditional table-based Q-Learning approximates $Q^*(s, a)$ recursively through successive Bellman updates,

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha\big(y_t - Q(s_{t+1}, a)\big) \tag{4}$$

with y_t being the Temporal Difference(TD) target for the Q-function.

$$y_t = R_t + \gamma \max_{a_{t+1}} Q^{\pi}(s_{t+1}, a_{t+1}) \tag{5}$$

This table representation is not useful for high dimensionality cases, since the size of our table would increase exponentially, nor for continuous cases, since every distinct $s \in \mathcal{S}$ would require an entry.

C. Deep Q Network

One way of dealing with the issues of Q-Learning in high dimensional spaces is to use a Neural Network as function approximator, called Deep Q-Network (DQN) [5]. The Q-function approximation is denoted then in terms of the parameters θ of the DQN as $Q(s, a, \theta)$. One component of DQN that stabilises learning is the Target Network. DQN uses two neural networks: the main network with parameters θ, which approximates the Q-function, and the target network with parameters θ^- which provides the TD targets for the DQN updates and is updated every number of episodes by copying

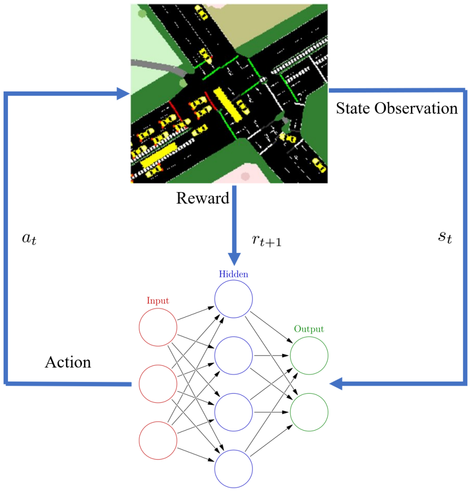

Fig. 1. Schematic representation of information transmission in Reinforcement Learning taking place between Agent and SUMO Environment

Fig. 2. Aerial view of Study Junction from Google Earth.

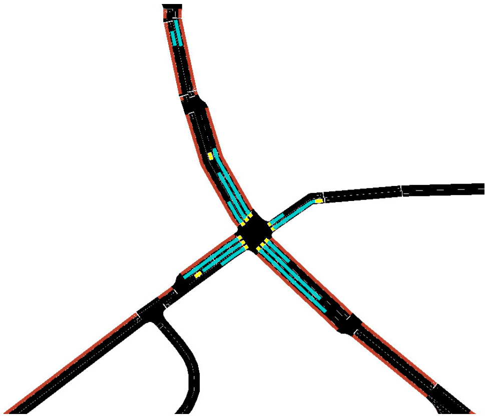

Fig. 3. Study Junction model in SUMO with a schematic representation of the areas covered by sensors.

the weights $\theta^- \leftarrow \theta$. With $Q^\pi(s_{t+1}, a_{t+1}, \theta^-)$ representing the target network, it results in a TD target to approximate:

$$y_t = R_t + \gamma \max_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1}, \theta^-) \tag{6}$$

IV. Methods

A. Reinforcement Learning Agent

The basic common agent used to obtain these results is a standard implementation of a DQN in PyTorch [8], optimising its weights via Stochastic Gradient Descent [9] using ADAM [10] as optimizer. The learning rate is $\alpha = 10^{-5}$ and the discount factor is $\gamma = 0.8$ for all simulations. The Neural Network in the agent uses 2 hidden, fully connected layers of sizes 500 and 1000 respectively, using ReLU as an activation function.

B. Reinforcement Learning Environment

The environment is modelled in the microscopic traffic simulator SUMO [2], representing a real-world intersection in Greater Manchester, UK. The junction consists of four arms, with 6 incoming lanes (two each in North-South orientation, and one each in East-West orientation) and 4 pedestrian crossings. The real-world site also contains 4 Vivacity vision-based sensors, able to supply queue length, speed and flow data. The demand and turning rations at the junction have been calibrated using 3.5 months of journey time and flow data collected by these sensors. The environment includes an emulated traffic signal controller, responsible of changing between the different stages in the intersection and enforcing the operational limitations, focused on safety. This includes enforcing green times, intergreen times, as well as determining allowed stages.

A stage is defined as a group of non-conflicting green lights (phases) in a junction which move at the same time.

The Agent decides which stage to select next and requests this from an emulated traffic signal controller, which moves to that stage subject to its limitations, which are primarily safety-related. The site features 4 Vivacity vision-based sensors which can provide flow, queue length and speed data. The data available to the agent is restricted to what can be obtained from these sensors.

C. State Representation

The agent receives an observation of the simulator state as input, which remains constant across all experiments here presented. This observation is a combination of the state of the traffic controller (which stage is active) and data from the sensors. The data from the sensors is comprised of the occupancy in each lane area detector and a binary signal representing whether the ”Pedestrian Cross” button has been pushed or not. A series of these, covering the last 12 seconds at a resolution of 0.6 seconds (20 state-entries) are provided to the agent.

While there is a variety of state representations available in the literature, often being more information dense, many of their features are not obtainable in practice with available sensors. Furthermore, recent studies [7] show that more information-dense states provide marginal improvements, if any, being able to control isolated intersections with simple inputs.

- 7 * Area sensor occupancies
- VDR3 Pedestrian Button Push
- G1-G4 Stage bits (Binary, active/inactive)
- NS1-NS4 Next Stage Bits (ASK SHAUN WHERE THEY COME FROM, FUNCTION AND HOW TO OBTAIN THEM)

D. *Actions of the Agent*

The actions available to the agent

The junction is configured to have 4 available stages. The Agent is able to choose Stage 2, Stage 3 or Stage 4, yielding an action space size of 3. Stage 1 serves a leading right turn phase from the main road, and was excluded by suggestion of the transport authority, since it is an intermediary stage that the controller will go through in order to reach Stage 2, which serves the main road. Stage 3 only serves pedestrians, which are not considered here, so was also excluded. Stage 4 serves the side roads, which do experience significant demand. In each timestep when a stage has been active longer than the minimum green time, the agent generates state-action values for each potential stage and the highest value is chosen according to an ϵ-greedy policy [6]. If the agent chooses the same stage, that stage is extended by 0.6s, otherwise the controler begins the transition to the other stage. The extension can be chosen indefinitely, as long as the agent identifies it as the best action.

The complexity in the decision-making stems from the combination of using of Stage 1 as an intermediate state and the extensions to the stage duration. Traditional RL for Urban Traffic Control regards each Stage as an action for the agent to take, based on the instantaneous state of the system. However, in the case of the intermediate Stage 1, the agent has to choose when to start the transition without knowledge of the future state when Stage 2 begins. Regarding the extensions, given that their length is smaller than that of the initial phase, their impact on the state will be smaller, generating a distribution of reward and state-action value outcomes that the agent needs to approximate.

E. *Modal Prioritisation*

The agent has to serve vehicles and pedestrians arriving at the intersection, seeking to jointly optimise the intersection for both modes of transport.

All the reward functions presented in this paper follow the same structure. The reward, as seen by the agent, will be a linear combination of an independently calculated reward for the vehicles and another for the agents, as it can be seen in Eq. 7.

$$R_t = \alpha * R_t^v + \beta * R_t^p \tag{7}$$

In this way, α and β are the Modal Prioritisation coefficients for our rewards.

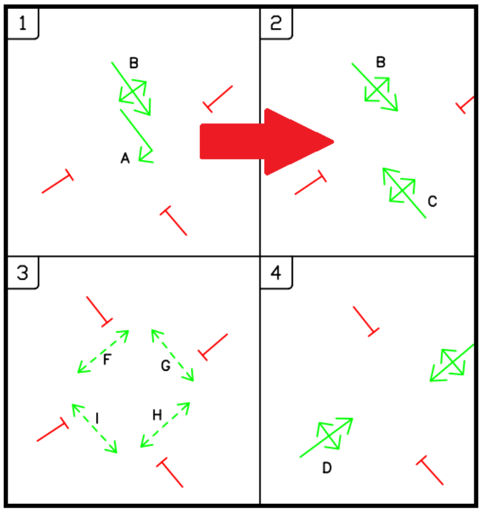

Fig. 4. Allowed stages and the phases that compose them. Stage 1 is an intermediate Stage, being necessary to go through it to reach Stage 2.

V. Reward Functions

In this section the individually tested reward functions are introduced.

Let N be the set of lane queue sensors present in the intersection. Let M be the set of pedestrian occupancy sensors in the junction. Let V_t and P_t be respectively the set of vehicles on incoming lanes, and the set of pedestrians waiting to cross in the intersection at time t. Let s_v be the individual speeds of the vehicles, τ^v and τ^p the waiting times of vehicles and pedestrians respectively. Let ρ_v and ρ_p the vehicular and pedestrian flows across the junction over the length of the action. Let t^p be the time at which the previous action was taken and t^{pp} the time of the action before that. Lastly, let t_e^v and t_e^p the entry times of vehicles and pedestrians to the area covered by sensors.

A. *Queue Length based Rewards*

1) *Queue Length:* The reward will be the negative sum over all n sensors of the queues (q) at time step t. Similar to the reward introduced in [**?**] but without the need for thresholding the queue values, and used in [**?**].

$$R_t = -\alpha \sum_{n \in N} q_t^v - \beta \sum_{m \in M} q_t^p \tag{8}$$

One of the first published in the field of Q-Learning, this reward function has low sensor requirements and is implementable using just induction loops.

2) *Queue Squared:* Introduced in [**?**], this function squares the result of adding all queues. This increasingly penalises actions that lead to longer queues.

$$R_t = -\alpha \left(\sum_{n \in N} q_t^v \right)^2 - \beta \left(\sum_{m \in M} q_t^p \right)^2 \tag{9}$$

3) Delta Queue: The reward will be the difference between the previous and current sum of queues, turning positive for those action that decrease the queue size and negative when it increases. Similar approach to the rewards shown in Eqs. (12) and (15).

$$R_t = \alpha\Big(\sum_{n\in N} q^v_{t^p} - \sum_{n\in N} q^v_t\Big) + \beta\Big(\sum_{m\in M} q^p_{t^p} - \sum_{m\in M} q^p_t\Big) \quad (10)$$

B. Waiting Time based Rewards

1) Wait Time: The reward will be the negative aggregated time in queue (τ) that the vehicles at the intersection have accumulated since the last action.

$$R_t = -\Big(\sum_{v\in V_t} \tau^v_t + \sum_{p\in P_t} \tau^p_t\Big) \quad (11)$$

This function is more information-dense than queues, scaling with individual waiting times, but requires more advanced hardware for individual vehicle recognition.

2) Delta Wait Time: Similar to Eq. 10, used in [**?**]. The reward will be the difference between total intersection τ between the current time and the previous action.

$$R_t = \alpha\Big(\sum_{v\in V_t} \tau^v_{t_p} - \sum_{v\in V_t} \tau^v_t\Big) + \beta\Big(\sum_{p\in P_t} \tau^p_{t_p} - \sum_{p\in P_t} \tau^p_t\Big) \quad (12)$$

3) Waiting Time Adjusted by Demand: The reward will be the negative aggregated waiting time as above, but in this case it is divided by an estimate of the current demand ($\hat{d}$) or arrival rate, implicitly accepting that given a wait time as a result of an action, the penalty should scale with the difficulty of the task.

$$R_t = -\frac{1}{\hat{d}}\Big(\alpha\sum_{v\in V_t} \tau^v_t + \beta\sum_{p\in P_t} \tau^p_t\Big) \quad (13)$$

C. Time Lost based Rewards

1) Time Lost: Used in [**?**], the reward will be the negative aggregated delay accumulated by all vehicles upstream from the intersection, understanding the delay as deviations from the vehicle's maximum allowed speed (s_{max}). Assuming a simulator time step of length δ:

$$R_t = -\Big(\alpha\sum_{v\in V_t}\sum_{t^p_e}^{t} \delta * (1 - \frac{s_v}{s_{max}}) + \beta\sum_{p\in P_t} \tau^p_t\Big) \quad (14)$$

This reward provides a more accurate representation of the total delay caused, since it also accounts for all deceleration happening around the intersection.

2) Delta Time Lost: Introduced in [**?**] and used in [**?**] [**?**] [**?**] [**?**] and [7]. Similar to Eq. (12). The reward will be the change of global delay in the vehicles around the intersection since the last action was taken.

$$R_t = \alpha\Big(\sum_{v\in V_t}\sum_{t^{pp}}^{t^p} ts{*}(1 - \frac{s_v}{s_{max}}) - \sum_{v\in V_t}\sum_{t^p}^{t} ts * (1 - \frac{s_v}{s_{max}})\Big)$$
$$+ \beta\Big(\sum_{p\in P_t} \tau^p_{t^p} - \sum_{p\in P_t} \tau^p_t\Big) \quad (15)$$

This reward function provides both punishment and reward centered around zero.

3) Delay Adjusted by Demand: The reward will be the same as in the point above, but divided by an estimate of the demand level.

$$R_t = -\frac{1}{\hat{d}}\Big(\alpha\sum_{v\in V_t}\sum_{t^p}^{t} ts * (1 - \frac{s_v}{s_{max}}) + \beta\sum_{p\in P_t} \tau^p_t\Big) \quad (16)$$

D. Average Speed based Rewards

1) Average Speed, Wait Time Variant: This reward seeks to maximise the average joint speed of all vehicles around an area of influence around the intersection. Let τ_{max} be a maximum desirable pedestrian waiting time.

$$R_t = \alpha\frac{\sum_{v\in V_t}\frac{s}{s_{max}}}{\sum_{V_t} v} + \beta\sum_{p\in P_t}\frac{\tau^p_t}{\tau_{max}} \quad (17)$$

2) Average Speed, Occupancy Variant: This reward seeks to maximise the average joint speed of all vehicles around an area of influence around the intersection. Let p_{max} be a maximum desirable number of pedestrians in the crossing.

$$R_t = \alpha\frac{\sum_{v\in V_t}\frac{s}{s_{max}}}{\sum_{V_t} v} + \beta\sum_{p\in P_t}\frac{p}{p_{max}} \quad (18)$$

3) Average Speed Scaled by Demand, Demand and Occupancy: The reward will be, as in the previous two entries, but multiplied by an estimation of the demand $\hat{d}$. This function scales the reward with the difficulty of the task.

E. Throughput based Rewards

1) Throughput: The reward will be the total number of vehicles that cleared the intersection between the last time that an action was taken and now.

$$r_t = \alpha\sum_{t_p}^{t}\rho_v + \beta\sum_{t_p}^{t}\rho_p \quad (19)$$

VI. Experiments

A. DQN Agents Training

The training process covered 1500 episodes running for 3000 steps of length $\delta = 0.6$ seconds for a simulated time of 30 minutes (1800 seconds). The traffic demand is increased as the training advances, with the agent progressively facing sub-saturated, near-saturated and over-saturated scenarios, with a minimum of 1 vehicle/3 seconds (1200 vehicles/h) and a maximum of 1 vehicle/1.4 seconds (2571 vehicles/h).

For each reward function, 10 copies of the agent were trained, and their performance was compared against two reference systems. These were Maximum Occupancy (longest queue first) and Vehicle Actuated System D [11], commonly used in the UK. The agent performing best in each class was selected for scoring.

B. Evaluation and Scoring

Each agent is tested and its performance scored over 100 copies of 3 different scenarios with different demand levels. Each evaluation has the same length as the training episodes, and the demand is kept constant during each run. These 3 scenarios are aimed to test the agents during normal operation, peak times and over-saturated conditions, and will be henceforth referred to as Normal, Peak and OverSat. Peak Scenario uses the level of demand observed in the junction that results in saturated traffic conditions under traditional controllers.

The Normal Scenario uses an arrival rate of 1 vehicle / 2.1 seconds (1714 vehicles/h). Peak Scenario uses an arrival rate of 1 vehicle / 1.7 seconds (2117 vehicles/h). OverSat Scenario uses an arrival rate of 1 vehicle / 1.4 seconds (2400 vehicles/h)

VII. Results

VIII. Discussion and Conclusion

Methods based on maximisation of the average speed seem to be Pareto Optimal in this context. In them, the travel times are allocated in a way, such that attempting to optimise any of our result variables (i.e. vehicle travel time and pedestrian travel time) will have a negative impact on the other one.

Acknowledgment

This work was part funded by EPSRC Grant EP/L015374 and part funded by InnovateUK grant 104219. We are also grateful to W. Chernikoff, Toyota Mobility Foundation and The Alan Turing Institute for support in the initial stages.

References

[1] Haydari, A., & Yilmaz, Y. (2020). Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey. arXiv preprint arXiv:2005.00935.

[2] Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y. P., Hilbrich, R., ... & WieBner, E. (2018, November). Microscopic traffic simulation using sumo. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC) (pp. 2575-2582). IEEE.

[3] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

[4] Melo, F. S. (2001). Convergence of Q-learning: A simple proof. Institute Of Systems and Robotics, Tech. Rep, 1-4.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[6] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[7] Genders, W., & Razavi, S. (2018). Evaluating reinforcement learning state representations for adaptive traffic signal control. Procedia computer science, 130, 26-33.

[8] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Desmaison, A. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (pp. 8024-8035).

[9] Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics, 23(3), 462-466.

[10] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[11] Highways Agency (2002). Siting Of Inductive Loops For Vehicle Detecting Equipments At Permanent Road Traffic Signal Installations. MCE 0108 Issue C.