Supplementary Material for IG-RL

François-Xavier Devailly, Denis Larocque, Laurent Charlin

1 Time complexity

In this section, we discuss time complexity of IG-RL with respect to :

1. Preprocessing states and building the computational graph required to perform convolutions.
2. Training & Evaluation under the settings described in the manuscript
3. Scaling & Transfer

1.1 Building the computational graph

The computational graph is the preprocessed/network representation of the state of the environment at a given timestep. It is on this graph that convolutions are performed to obtain Q-values. There are two 'parts' in the computational graph: The 1st part is the fixed part (the architecture of the road network which does not evolve). It includes TSC nodes, connection nodes, lane nodes, and the lane length feature for all lanes. This part is only built once per road network (no need to update it). The 2nd part is the dynamic part. It includes vehicle nodes, and most node features (§ IV-B : IG-RL – Model). This is the part that is updated every timestep.

1.2 Training & Evaluation

To run the experiments presented in the manuscript, we used 2 GPUs (RTX 2080Ti, Memory : 11GB) for training and 1 for evaluation, and a single CPU (AMD Ryzen Threadripper 2950X) for both training and evaluation.

1.2.1 Synthetic road networks

As mentioned in § V-A8 : Experiments - General Setup - Robustness, to ensure the robustness of our conclusions, every experiment, including both training and evaluation, is repeated 5 times using different seeds. We now provide average estimates (over these 5 instances) of the required time to complete training and evaluation steps for IG-RL (once).

Table 1: Durations on synthetic road networks

Architecture	IG-RL-L		IG-RL-V
Training	Specialist	Generalist	
mean (min)	69.54	66.97	79.6
std dev	6.07	0.48	1.09

Table 2: Training (5 runs)

Architecture	IG-RL-L		IG-RL-V
Training	Specialist	Generalist	
mean (min)	4.29	4.27	5.24
std dev	0.38	0.40	0.66

Table 3: Evaluation (5 runs)

IG-RL-L Vs. IG-RL-V

- IG-RL-L does not use vehicle nodes. For this reason, the graph will only be built once (all nodes are fixed) and only features will need to be updated between timesteps. Message-passing complexity is constant between timesteps (does not vary with demand) for this version, hence the lower training and evaluation durations.
- IG-RL-V uses vehicle nodes. For this, reason, the graph needs to be updated at every timestep. Features must also be updated. Message-passing complexity increases linearly with the number of vehicles (between timesteps) for this version, hence the longer training and evaluation durations.

Generalist Vs. Specialist The training duration of G-IG-RL(L and V) has relatively low variability because of the variety of road network architectures used to train it. In comparison, the single road network architecture used for S-IG-RL can have a significant impact on training duration.

1.2.2 Manhattan road network

To build the Manhattan road network and all included objects, we imported all road network data/details from OpenStreetMap on SUMO. We generate traffic/demand using the same method that is described in the 1st experiment (§V-A4: Experiments - General Setup – Traffic Generation)

For the corresponding experiments, we completed all 5 evaluation runs simultaneously (in parallel). Completing the evaluation scenario described in Experiment 2 for G-IG-RL-L took ∼23 hours and 30 minutes.

1.3 Scaling & Transfer

1.3.1 Scaling (number of intersections)

Every intersection operates independently (in a decentralized fashion). Both the construction of the computational graph and the message-passing can be performed independently as well. Consequently:

- The total computational cost of both the construction of the computational graph and message passing scales linearly with the number of intersections (assuming intersections have architectures of comparable complexity).
- Control of all TSCs can be parallelized (performed independently).

1.3.2 Transfer complexity

Where other methods would require to re-train from scratch on any new intersection and/or road network, requiring many experiences and substantial training for any new setting, one of the main advantage of G-IG-RL is the ability to make training time independent of the applications of the model after it is trained.

2 Pseudo-code

In this section, we provide a pseudo-code which describes how the gathering of experiences and the training of the model are orchestrated.

2.1 Interaction with SUMO

SUMO's Traffic Control Interface (TraCI) enables retrieving values of simulated objects and manipulating their behavior "on-line". Simulation processes run multiple instances of SUMO in parallel and are used to both 1) gather experiences for training and 2) perform evaluation.

At every timestep t of a given episode (on a given simulation process), data retrieved using TraCI constitutes the current state (s_t) of the environment/road network (and therefore the state of every included MDP). This data is preprocessed into a network G_{s_t} (nodes, node embeddings/features, and edges) on which we perform graph convolutions to obtain the Q-values for every TSC (§ IV : IG-RL). For every TSC, the action with the highest predicted Q-value is selected and fed back to SUMO to compute/obtain the next state. Exploration is enabled by the use of a noisy fully connected layer (§ IV : IG-RL).

Algorithm 1: Training IG-RL (with double Q-Learning)

Input: M : Total number of steps per simulation process
Input: T : Episode duration
Input: J : Number of training iterations per training epoch
Input: N : Batch size
Initialize Q-Network (GCN + noisy fully connected layer) with random weights θ
Initialize target Q-Network with $\theta^{'} = \theta$
Initialize replay buffer R
do in parallel
- **Simulation processes (gathering experiences by interacting with SUMO) :**
  - m=0
  - **while** $m<M$ **do**
    - Generate and start episode (road network and trips)
    - Build the root computational graph G_{root} corresponding to the empty road network (§ 1.1)
    - Observe initial state s_1
    - **for** $t=1,T$ **do**
      - Update G_{root} to obtain the current computational graph G_{s_t} (§ 1.1)
      - Select action $a_t = argmax_a Q^{\theta}(G_{s_t}, a)$
      - Execute action a_t, observe reward r_t and new state s_{t+1}
      - m = m + 1
    - Send all transitions from the episode $(G_{s_t}, a_t, r_t, G_{s_{t+1}})$ to the training process
  - Terminate simulation process
- **Training process (training the Q-Network) :**
  - **while** *simulation processes are not all terminated* **do**
    - **for** *each simulation process* **do**
      - Verify if new transitions are available and store them in R
    - **for** $j=1,J$ **do**
      - Sample a random minibatch of N transitions from R
      - Set double Q-Learning target, $y_i = r_i + \gamma Q^{\theta}(G_{s_{t+1}}, argmax_{a'} Q^{\theta^{'}}(G_{s_{t+1}}, a'))$
      - Update Q-Network (θ) by minimizing the loss : $L = \frac{1}{N} \sum (y_i - Q^{\theta}(G_{s_t}, a_t))$
    - Update target Q-Network with $\theta^{'} = \theta$
    - Send updated θ to simulation processes
  - Save Q-Network (θ)

3 Experiment 1 : Delay Evolution (full version)

For clarity, Fig. 5 (in the manuscript) focuses on competitive approaches with lower delays (which stabilize early on). In this section, we provide the full version of this figure.

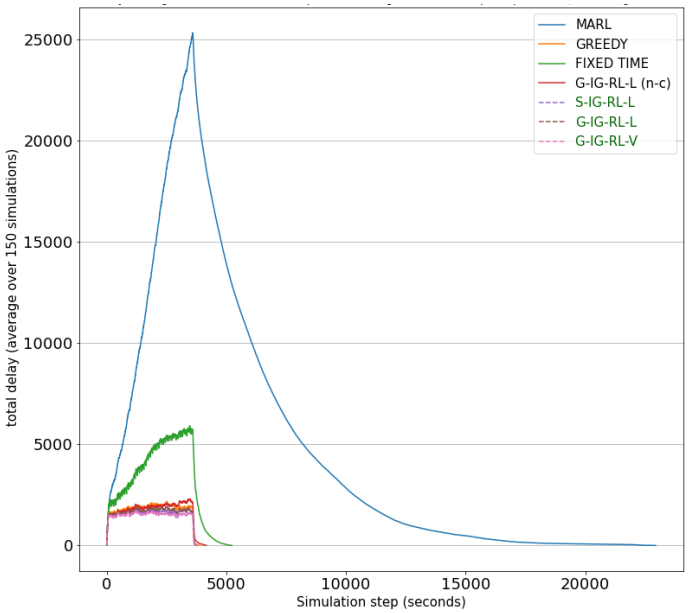

Figure 1: Total Delay Evolution: **Default** Traffic Regime — Synthetic Road Networks. As described in § VD-2e, we see that MARL's lack of generalization ability under the shift in traffic distribution between training and evaluation settings causes catastrophic performance (i.e., massive delays). Fixed-Time is also performing relatively bad compared to competitive methods but seems to have started converging after an hour.