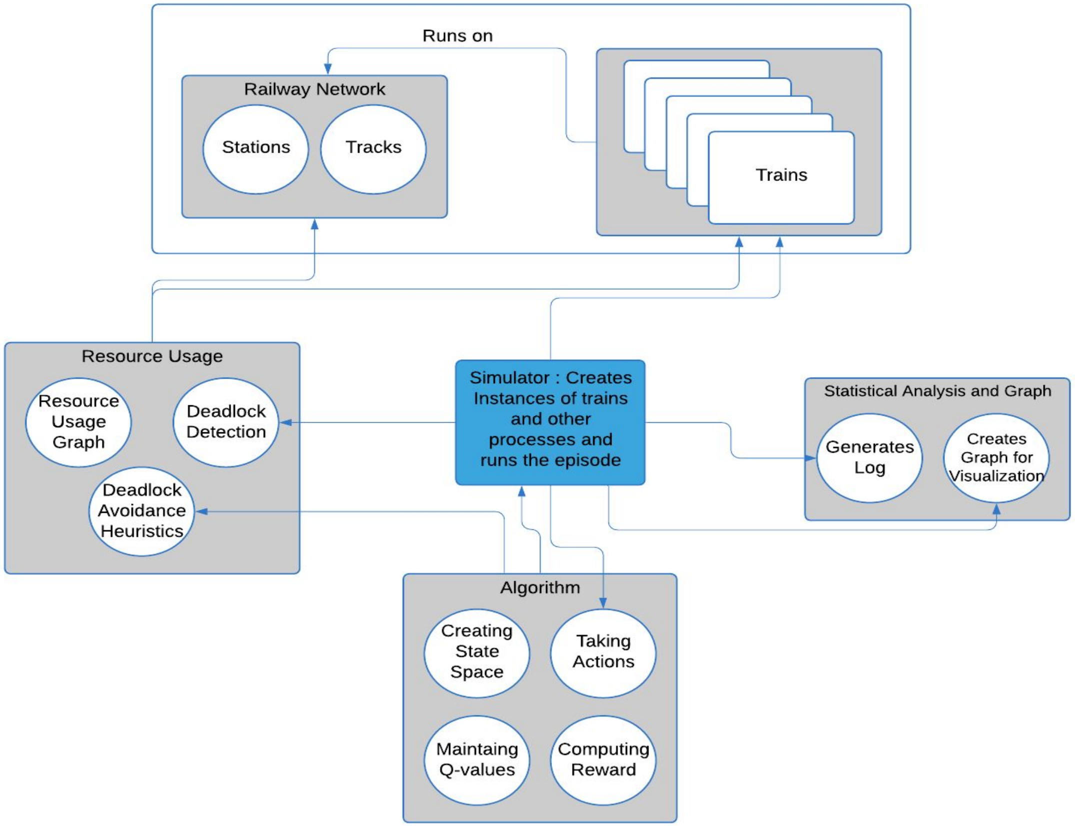
Railway Scheduling using Reinforcement Learning

Arpit Singh
111601031

Scheduling Problem

- **Resources** - Network topology, Stations, Tracks.
- **Train Movement** - Reference timetable (desired arrival and departure time of each train at each station)
- **Goal** - Assign track resources for each train for a fixed time period, such that they all complete their journeys without conflicts.
- Timetable may be **infeasible**.
    - Adjust arrival and departure times such that all rules are satisfied, while minimizing **Priority Weighted delay**.
- **Rescheduling** (online counterpart)
    - **Goal** : Recover from a disruption of timetable (due to delays or faults)

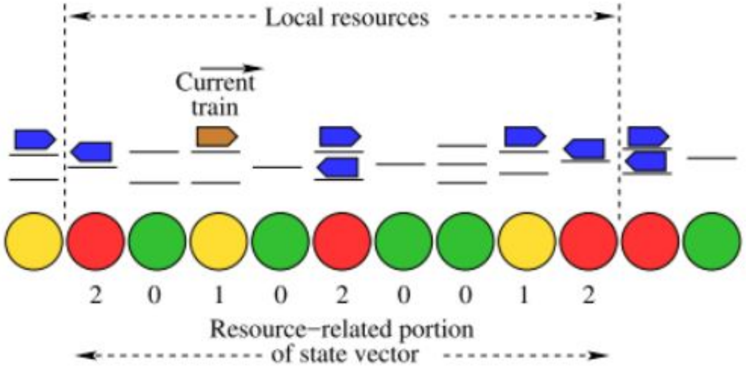

Runs on
Railway Network
Stations
Tracks
Trains
Resource Usage
Resource Usage Graph
Deadlock Detection
Deadlock Avoidance Heuristics
Simulator : Creates Instances of trains and other processes and runs the episode
Statistical Analysis and Graph
Generates Log
Creates Graph for Visualization
Algorithm
Creating State Space
Taking Actions
Maintaing Q-values
Computing Reward

Overview

- Algorithmic Details
  - Sarsa (λ)
  - Proxy Reward
  - Prior and Proposed Q-values
- Experiments on (HYP-1 , HYP-2 , HYP-3)
- Results and Insights (Transfer Learning)
- Flatland Challenge
- Future Work

State Space Representation

- Local neighborhood
- Higher values indicate higher congestion in the resource
- l_b behind each train and l_f infront of each train
- $S_r = R - 1 - min(R - 1, \lceil N_r - w_c T_{r,c} - w_d T_{r,d})$
- Priority of the train is also in state
- Size of state space $P * R^{l_b+1+l_f}$



Action and Policy Definition

- Choice of action in any given state is **binary**
  - **Move** - to move the current train to the next resource
  - **Wait** - halt in the current resource for a predefined time period
- The order in which trains are selected is given be **deadlock avoidance heuristic**
- **ϵ - greedy policy** based on Q-value
  - With probability ϵ, choose action randomly (**Exploration**)
  - With probability (1-ϵ) , choose action greedly (**Exploitation**)
- **ϵ -** decreases as the training proceeds

Objective Function

- Time duration from first event to last event (makespan)
- Total or average running time of trains
- Robustness of the timetable to deviations (**using Slack times**)
- Priority-weighted delay

$$J = \frac{1}{N_{r,t}} \sum_{r,t} \frac{\delta_{r,t}}{P_t}$$

Sarsa(λ)

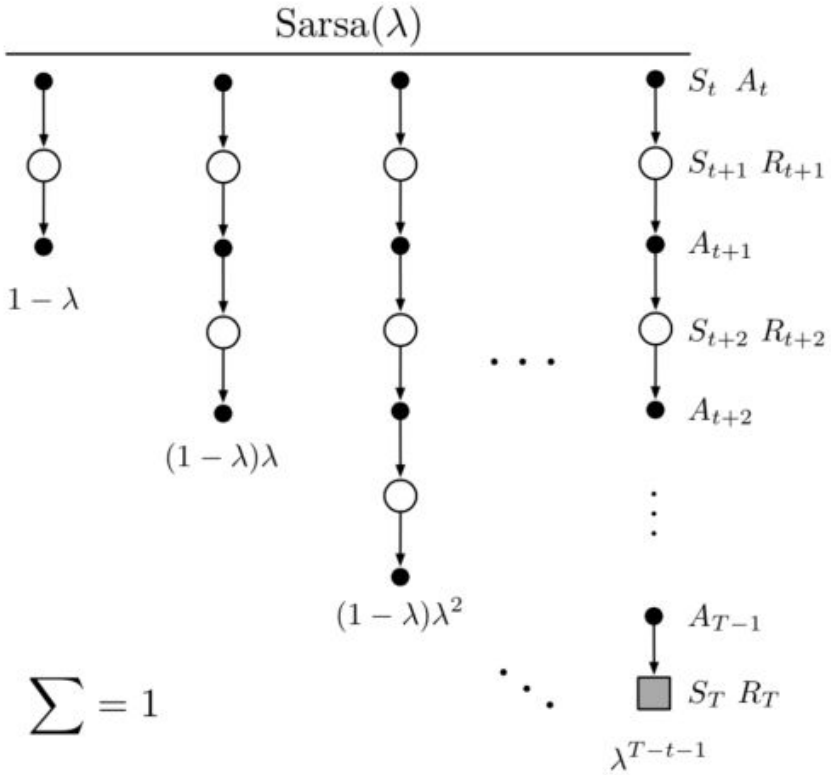
- **Objective function** as the negative of the reward.
- Reward at the end of each episode

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \quad \forall(s,a)$$

$$\delta_t = r_{t+1} + \gamma * Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & if\ s = s_t \ \ and\ a = a_t \\ \gamma\lambda e_{t-1}(s,a) & otherwise. \end{cases}$$

Sarsa(λ) backup diagram



Sarsa(λ) Algorithm

Algorithm 1 Sarsa Lambda [3]

Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0\ \forall s, a$
Repeat (for each episode):
  Initialize s,a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using ϵ - greedy policy
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    For all s,a
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
      $e(s, a) \leftarrow \gamma\lambda e(s, a)$
    $s \leftarrow s'; a \leftarrow a'$
  until s is terminal

About Problem Instances

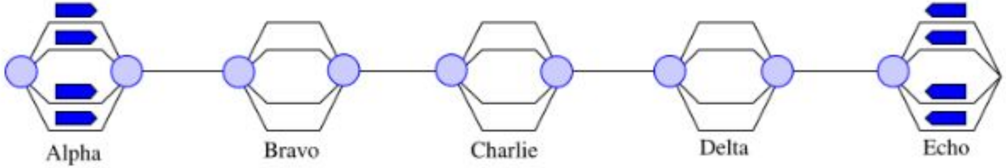
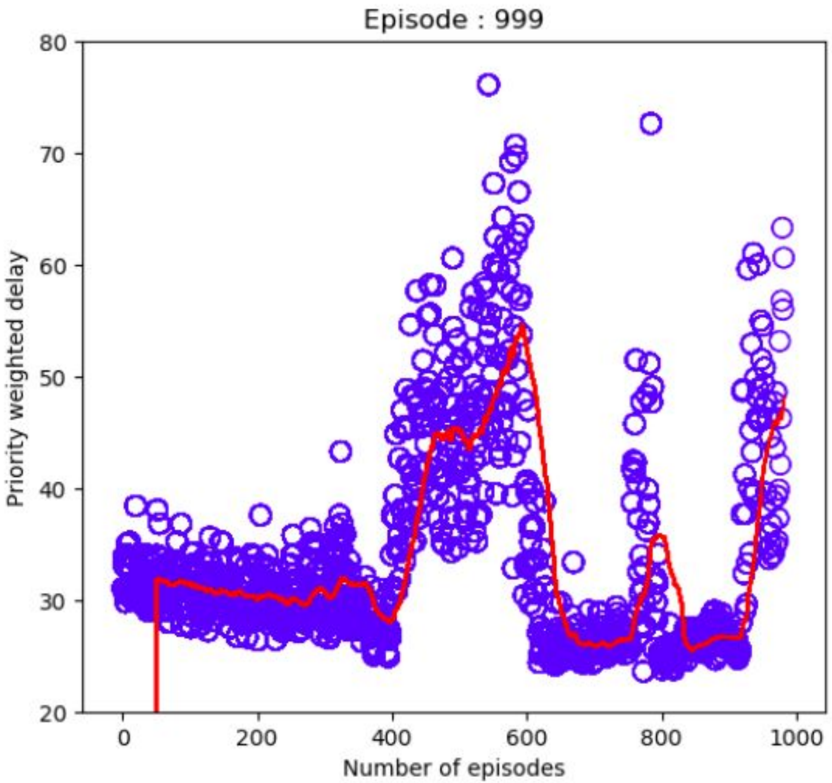


Table 5.1 Hypothetical instances

Name	Stns.	Trains (sorted by priority)	Events
HYP-1	5	8,0,0	40
HYP-2	11	15,45,0	1320
HYP-3	11	40,80,0	2640

Sarsa(λ) Result

- The back-propagation of rewards after the end of the episode is not possible, because the episode can be very long.
- Possible to visit the same state-action pair in loop leading to large accumulation of reward at that state-action pair, leading to extreme values.
- the magnitude of delays is different from one problem instance to another (obstacle in transfer learning).



Proxy Reward

- **Success** of episode
  - Maximum acceptable level J is set to a proportion (1 + ρ) of the minimum J observed thus far.
  - **Success** : If sum of priority-weighted delay is under current threshold, reward +1
  - **Failure** : if priority weighted delay is over the threshold, or if the episode enters deadlock, reward 0
- Instead of tracking whole trajectory, observe what state-action pairs are visited in an episode.
- **Proxy Reward** : For a give state-action pair, probability of ending up in a successful episode

$$0 \leq \sigma(x, a) = \frac{\epsilon^*_{x,a}}{\epsilon_{x,a}} \leq 1$$

Q-values

Prior

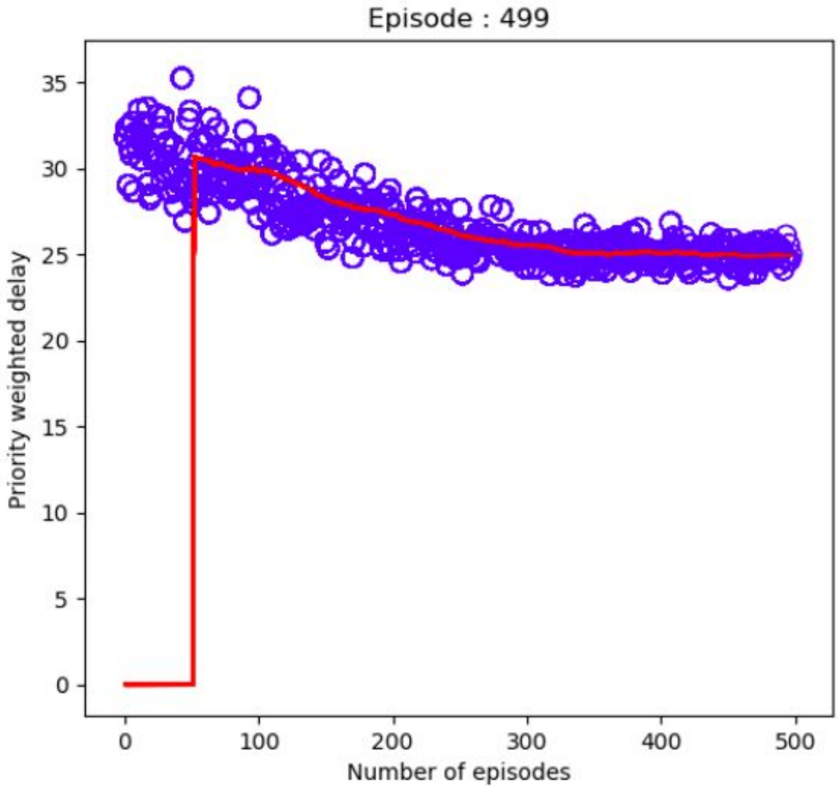
$$q(x,a) = w\sigma(x,a) + (1-w)\sum_{m=1}^{M}\frac{\sigma(x'_m, a'_m)}{M}$$

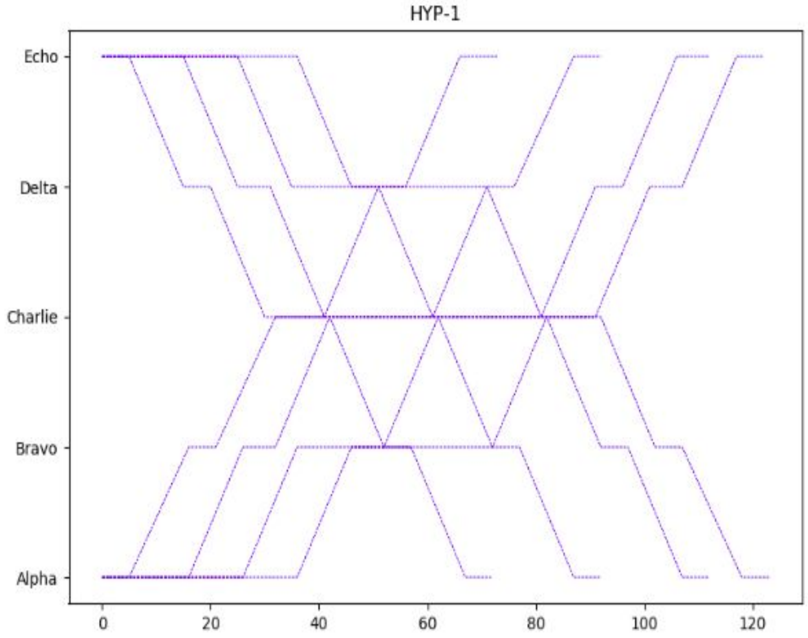
Propose

$$q(x,a) = w\sigma(x,a) + (1-w)\sum_{m=1}^{M}\frac{q(x'_m, a'_m)}{M}$$

$$q(x,a) = \sigma(x,a) + \gamma\sum_{m=1}^{M}\frac{q(x'_m, a'_m)}{M}$$

Training (HYP-1)



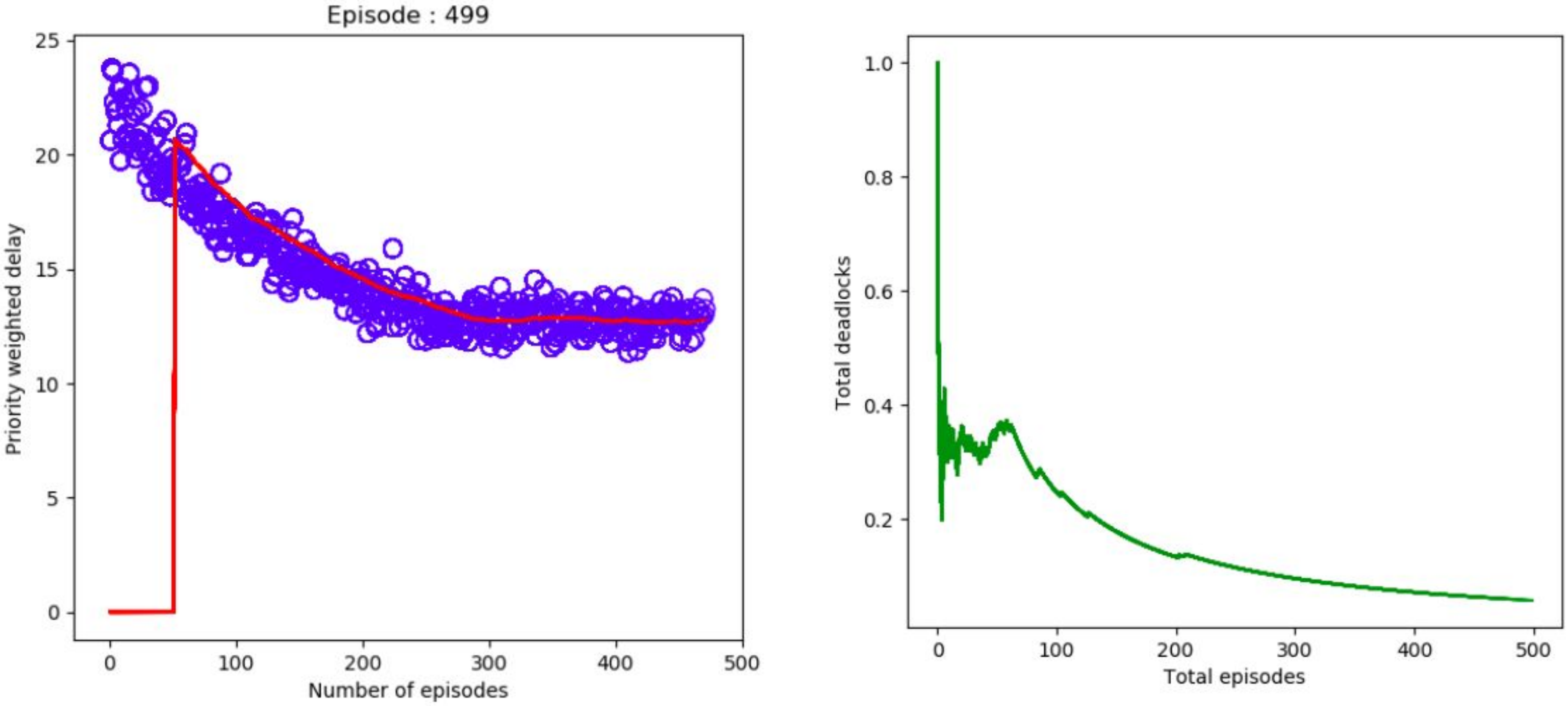


Training

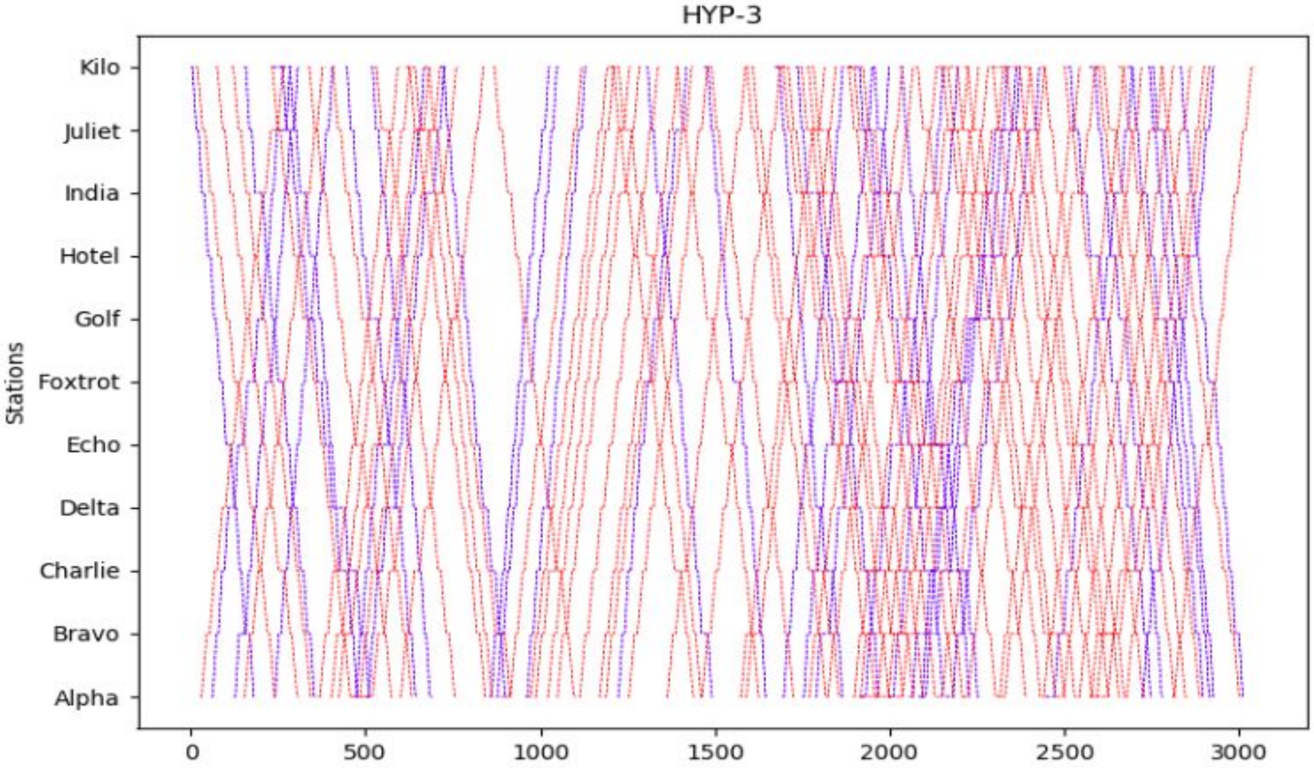
Table 5.3 Training

Instance	Minimum	Deadlock	Total states visited
HYP-1	23.53750	3	354
	23.58750	3	348
HYP-2	2.60682	3	1650
	2.58447	5	1712
HYP-3	11.64754	32	3377
	11.34754	29	3021

Training (HYP-3)



HYP-3 Schedule



Testing (Zero delay)

Table 5.4 Zero delay

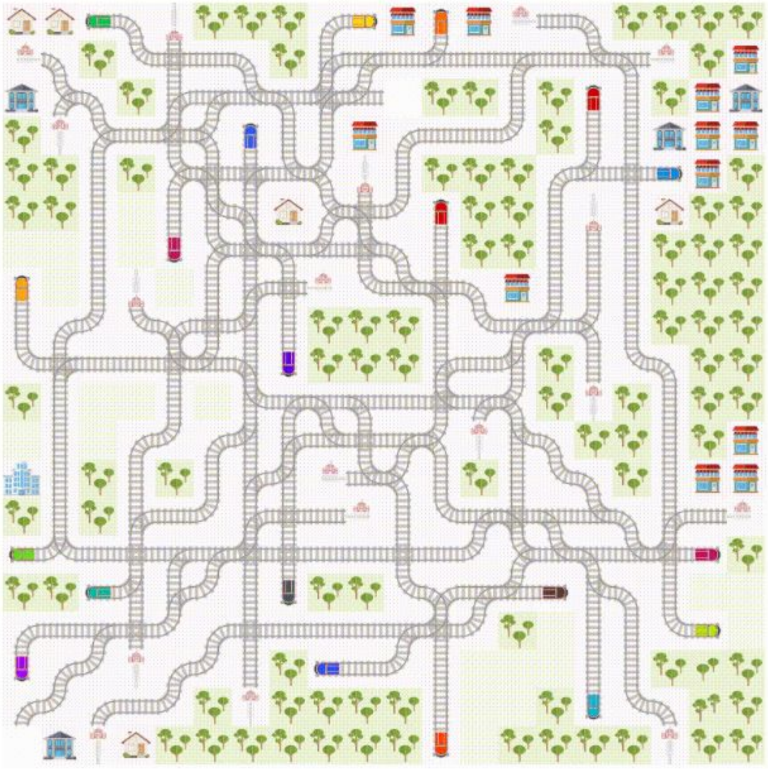
Train	Test	Minimum	Average	deadlock
HYP-2	HYP-2	4.050	4.980 ± 0.590	0
		2.680	$\mathbf{3.257 \pm 0.501}$	**0**
HYP-3	HYP-2	3.089	$\mathbf{4.080 \pm 0.370}$	0
		2.709	4.184 ± 0.438	**0**
HYP-2	HYP-3	12.683	14.580 ± 1.058	16
		11.453	$\mathbf{13.083 \pm 1.164}$	**6**
HYP-3	HYP-3	11.855	12.954 ± 0.540	1
		11.438	$\mathbf{12.734 \pm 0.613}$	**0**

Testing (20% delay)

Table 5.5 20% delay

Train	Test	Minimum	Average	deadlock
HYP-2	HYP-2	9.591	11.388 ± 1.258	**0**
		8.386	$\mathbf{10.261 \pm 0.733}$	1
HYP-3	HYP-2	9.603	10.932 ± 0.881	3
		8.473	$\mathbf{10.472 \pm 0.792}$	**1**
HYP-2	HYP-3	**26.882**	31.955 ± 2.290	**23**
		27.734	$\mathbf{30.141 \pm 1.486}$	29
HYP-3	HYP-3	**26.522**	30.135 ± 1.649	3
		26.560	$\mathbf{29.231 \pm 1.723}$	**1**

Flatland Challenge



Future Work

- **Further Testing**
  - On **Real life dataset** (Ajmer and Konkan railway lines)
  - Introduce only in some part of the railway network
- Current algorithm works only for **railway line** not **railway network.** Try to extend current algorithm for railway network.
- Use **Tree like Observation** instead of Straight line.
- Work on solving Flatland challenge.