Flatland

Multi-Agent Reinforcement Learning for train scheduling

Lorenzo Borelli
Giovanni Minelli
Lorenzo Turrini

Deep learning course final project

06/10/21

How?

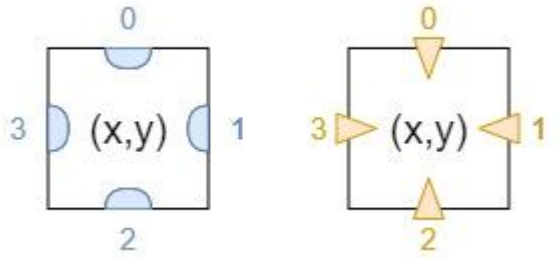
- Environment
- Challenge known approaches
- Colleagues experiments and results
- Ideas and intuitions
- implementations and literature approaches

Observation objectives

- Ease the search of the shortest path
- Retrieve with low effort information from the map
- Limit as possible recurrent computations for each agent
- Build a data structure simple and without ambiguity to feed the RL model with

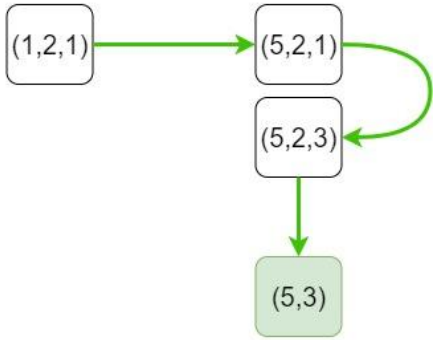
Separate the actual search of the minimum path from the detection of the feasible way

General graph



Expressive but concise, and built on the fly with low effort

Directed graph



Deadlock controller

We have developed two controller

- **Standard deadlock controller**
- **Observation deadlock controller**

Standard Deadlock controller

it uses a distance matrix of the environment but it does not use the information of observation.

PRO

it is independent of implementation of the observation.

CONS

- it is less precise for the same cost
- it does not have the control for the starvation

Observation Deadlock controller

it is based on dag observation, we have marked nodes of observation graph with label :

- **Conflict:** there is a possibility to have a deadlock situation for the current agent taking the direction of that switch because there is at least another agent facing that switch.
- **Deadlock:** The node marked conflict, if this situation becomes reality the node mark in deadlock and the state of agents change in deadlock when the step to deadlock is equals a zero.
- **Starvation :** it is marked when agent don't arrive to destination because the edges where there is a deadlock are delete from graph.

Observation Deadlock controller

PRO

- it is more precise for the same cost.
- it have the control for the starvation.
- it allows you to predict cases at risk.

CONS

- it is not independent of implementation of the observation.

Rewards

Intuitively an agent has to learn two behaviors, not necessarily in this order:

- Reach his destination in the shortest time possible.
- Avoid collisions with other agents.

Rewards

To obtain these goals, we decided to rewards:

- Having reached the destination
- decreasing the distance to reach the target

To obtain these goals, we decided to penalize:

- Deadlock agent
- Starvation agent

GCN

- **Normalize** input graph for DQN and A2C models.
- Embedded vector representation of graph features: minimum **distance** from neighbours and **type** of node.
- **Update rule**: adjacency and degree matrices.

DQN/D3QN

- Deep Q-learning and convergence to **Bellman optimality**.
- The model: fully connected layers, ReLU, **Huber loss** and **Kaiming initialization**.
- Target model: **fixed Q-targets**
- Picking experiences to train with **Prioritized Experience replay**
- Dueling architecture: **advantage** and **value** functions

A2C

- An **on-policy** method: learn the policy.
- **Actor** network: the policy .
- **Critic** network: the **baseline**.

Conclusions

- Performance-wise similar to default implementation, through computationally cheaper.
- Room for improvement:
  - More tests with different combination of hyperparameters.
  - Spectral normalisation.
  - Graphical channels like observation.