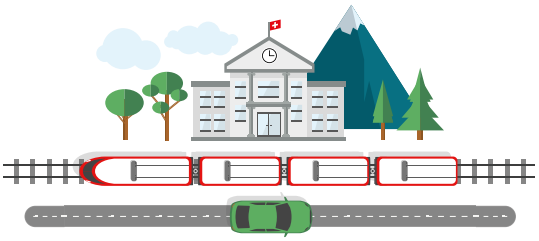
Alma Mater Studiorum · University of Bologna

Flatland Challenge



FLATLAND

Deep learning course final project

Leonardo Calbi (leonardo.calbi@studio.unibo.it)
Alessio Falai (alessio.falai@studio.unibo.it)

December 7, 2020

Contents

1 Introduction **4**

2 Background **5**

3 Environment **8**
3.1 Railway encoding . . . 8
3.2 Observations . . . 8
3.2.1 Tree . . . 8
3.2.2 Binary tree . . . 8
3.3 Predictions . . . 8
3.3.1 Shortest path . . . 8
3.4 Choices . . . 8
3.5 Rewards shaping . . . 8

4 Policy **9**
4.1 Action masking . . . 9
4.2 Action selection . . . 9
4.2.1 ϵ-greedy . . . 9
4.2.2 Boltzmann . . . 9
4.3 Replay buffers . . . 9
4.3.1 Uniform . . . 9
4.3.2 Prioritized . . . 9

5 DQN **10**
5.1 Architectures . . . 10
5.1.1 Vanilla . . . 10
5.1.2 Double . . . 10
5.1.3 Dueling . . . 10
5.2 Bellman equation . . . 10
5.2.1 Max . . . 10
5.2.2 Softmax . . . 10

6 GNN **11**

7 Results **12**

8 Conclusions **13**

List of Figures

2.1 Different `rail` cell types 5
2.2 Agents and targets 6
2.3 An example of a railway environment 6
2.4 Example of a `rail` cell transition matrix and bitmap 7

Foreword

The Flatland challenge is a competition organized by AIcrowd [1] with the help of SBB (Swiss Federal Railways) to foster innovation with what regards the scheduling of trains trajectories in a railway environment.

As reported on the official challenge website, SBB operates the densest mixed railway traffic in the world. It maintains and operates the biggest railway infrastructure in Switzerland. Today, there are more than 10 000 trains running each day, being routed over 13 000 switches and controlled by more than 32 000 signals.

The Flatland challenge aims to address the vehicle rescheduling problem by providing a simplistic grid world environment and allowing for diverse solution approaches. In particular, the first edition of the challenge was hosted during 2019 and the submitted solutions were mainly based on OR (Operation Research) methodologies, while the second edition of the competition, i.e. the NeurIPS 2020 edition, had the goal of favoring the implementation of RL (Reinforcement Learning) based solutions.

Introduction

At the core of this challenge lies the general vehicle re-scheduling problem (VRSP) proposed Li, Mirchandani, and Borenstein in 2007 [2]:

> *The vehicle rescheduling problem (VRSP) arises when a previously assigned trip is disrupted. A traffic accident, a medical emergency, or a breakdown of a vehicle are examples of possible disruptions that demand the rescheduling of vehicle trips. The VRSP can be approached as a dynamic version of the classical vehicle scheduling problem (VSP) where assignments are generated dynamically.*

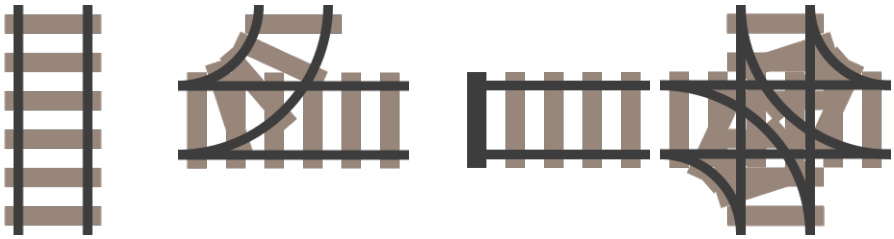
The problem is formulated as a 2D grid environment with restricted transitions between neighboring cells to represent railway networks. On the 2D grid, multiple agents with different objectives must collaborate to maximize the global reward.

The overall goal is to make all agents (trains) arrive at their target destination with a minimal travel time. In other words, we want to minimize the time steps (or wait time) that it takes for each agent in the group to reach its destination.

Background

As already pointed out, the Flatland environment is represented as a 2D grid and each cell can be one of many different types. The different types of cells can belong to the following categories: `rail` and `empty`.

The `rail` cells are the most intricated of the two, in that there exists different types of them. In particular, figure 2.1 shows examples of possible `rail` cells that can be used to build up a railway environment in Flatland. Other than the ones shown in figure 2.1 there are also diamond crossings (i.e. two orthogonal straight rails crossing each other), single slip switches (i.e. the same as double slip switches but with a single choice) and symmetrical switches (which are special kinds of switches that bifurcate to a left and right branch). Moreover, every `rail` cell can be rotated by 90° and mirrored along both axis, to allow more combinations between them to be made, in order to guarantee a great degree of diversity between different railways.



(a) Straight (b) Simple switch (c) Deadend (d) Double slip switch

Figure 2.1: Different `rail` cell types

Moreover, `rail` cells can be occupied by the following entities (the ones shown in figure 2.2):

- Agent: one `rail` cell can be seen as a resource with availability equal to one, so that in each time step only one agent can occupy it
- Target: they represent the destination of one or more agents (different agents could have the same target) and the number of possible targets present in the environment is clearly limited by the number of agents

An important fact about the different types of `rail` cells is that only switches require an agent to make a choice. In Flatland (like in reality) a maximum of

(a) Agent example (b) Target example

Figure 2.2: Agents and targets

two options is available. There does not exist a switch with three or more options.

Finally, every cell that is not `rail` is `empty` and neither targets nor agents can fill it up. As shown in figure 2.3, it is interesting to notice that Flatland is a very sparse environment, meaning that there are a lot more `empty` cells than `rail` ones: because of this, representing the environment as a simple dense matrix could lead to overheads and efficiency issues, especially when dealing with relatively big environments.

In the end, the only cell types that we care about are the `rail` ones: the `empty` ones are useful only for visualization purposes. Because of this, we could think of representing the environment as a sparse matrix or as a graph containing only `rail` cells or a subset of them (we will address this issue in chapter ??????????????????).

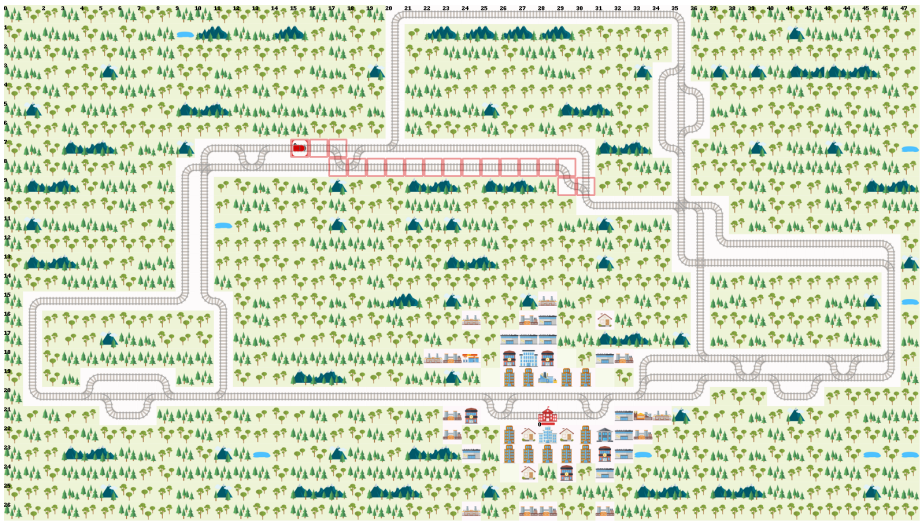


Figure 2.3: An example of a railway environment

An agent in the Flatland environment is a train that starts from a random `rail` cell in the map and has to arrive to its assigned `target` cell in the minimum number of steps. To do so, the agent can only occupy `rail` cells.

To move from a cell to another one the agent has to make a choice and,

depending on the cell type that they are on and on the connections between cells, an agent can transition from cell i, when looking towards direction d_i, to cell j, looking towards direction d_j, if and only if $T_i(d_i, d_j) = 1$, where T_i is the transition matrix associated to cell i, s.t. $T_i(d_i, d_j) = 0$ means that the transition from cell i, direction d_i to cell j, direction d_j is forbidden (likewise $T_i(d_i, d_j) = 1$ means that the transition is allowed). Directions d_* are represented as the 4 cardinal directions, i.e. North, East, South and West (N, E, S, W), so that each transition matrix T_* can be characterized as a 4×4 binary matrix. For example, a deadend cell like the one reported in figure 2.1c would have a transition matrix like the one shown in figure 2.4.

$$\begin{array}{c} \\ N \\ E \\ S \\ W \end{array} \begin{array}{c} \begin{array}{cccc} N & E & S & W \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \xrightarrow[\text{bitmap}]{\text{to}} 0000\ 000\mathbf{1}\ 0000\ 0000$$

Figure 2.4: Example of a `rail` cell transition matrix and bitmap

As we can observe, only one entry in the matrix has value 1, meaning that only one transition is possible, i.e. the one s.t. the agent enters heading East and exits heading West.

In the Flatland library, a transition matrix is represented by a bitmap, which can be seen as a linearization by rows of the reported matrix (the mapping between the transition matrix of the deadend cell 2.1c and its bitmap is again shown in figure 2.4). In this way, by simply counting the number of true values in the bitmap, we can understand the type of `rail` cell that we are examining (e.g. only one true value indicates a deadend, while exactly two true values indicate a straight rail).

Flatland has a discrete action space, meaning that only 5 possibilities have to be considered at each transition. In particular, Flatland uses the following convention:

1. `MOVE_FORWARD`: the agent maintains the current movement direction, if possible (i.e. if it was heading North, it will continue heading North)
2. `MOVE_LEFT`: if the agent is at a switch with a transition to its left, the agent will choose the left path, otherwise the action has no effect
3. `MOVE_RIGHT`: if the agent is at a switch with a transition to its right, the agent will choose the right path, otherwise the action has no effect
4. `STOP_MOVING`: the agent remains in the same cell
5. `DO_NOTHING`: the agent performs the same action as the last time step

Usually, only a handful of the reported actions can be perfomed on a given cell, meaning that most of the times the actual number of choices is much less than 5 (we will address this issue in chapter ??????????????????).

Environment

3.1 Railway encoding

3.2 Observations

3.2.1 Tree

3.2.2 Binary tree

3.3 Predictions

3.3.1 Shortest path

3.4 Choices

3.5 Rewards shaping

Policy

4.1 Action masking

4.2 Action selection

4.2.1 ϵ-greedy

4.2.2 Boltzmann

4.3 Replay buffers

4.3.1 Uniform

4.3.2 Prioritized

DQN

5.1 Architectures

5.1.1 Vanilla

5.1.2 Double

5.1.3 Dueling

5.2 Bellman equation

5.2.1 Max

5.2.2 Softmax

GNN

Results

Conclusions

Bibliography

[1] AIcrowd. *AIcrowd*. URL: https://www.aicrowd.com/.

[2] Jing-Quan Li, Pitu B. Mirchandani, and Denis Borenstein. "The vehicle rescheduling problem: Model and algorithms". In: *Networks* 50.3 (2007), pp. 211–229. DOI: https://doi.org/10.1002/net.20199.