UNIVERSITÀ DI BOLOGNA

School of Engineering

Deep Learning

Project 5: Flatland Challenge

Professor: **Andrea Asperti**

Students:
Giovanni Montanari
Lorenzo Sarti
Alessandro Sitta

Academic year 2019/2020

Abstract

The Flatland Challenge is a competition to foster progress in multi-agent reinforcement learning for any re-scheduling problem (RSP). The challenge addresses a real-world problem faced by many transportation and logistics companies around the world (such as the Swiss Federal Railways, SBB). Different tasks related to RSP on a simplified 2D multi-agent railway simulation must be solved: the proposed solutions may shape the way modern traffic management systems (TMS) are implemented not only in railway but also in other areas of transportation and logistics. This will be the first of a series of challenges related to re-scheduling and complex transportation systems [1].

Contents

1 Problem Description **5**

2 Proposed Solution **6**

3 Network model **8**

4 Single Agent **9**
4.1 Reducing distance reward 11
4.2 Standstill penalty 12

5 Multi Agent **14**
5.1 Training 3 agents: first attempt 15
5.2 Training 3 agents: deadlock penalty implementation 16
5.3 Training 3 agents: stop on switch penalty implementation . . 17
5.4 Training 5 agents: first attempt 18
5.5 Training 5 agents: mixed approaches 19
5.6 Training 10 agents: best result 20
5.7 Scalability 21
5.8 Considerations about agents' action distribution 21

6 Final Result **23**

Conclusions **26**

References **27**

Chapter 1

Problem Description

The Swiss Federal Railways (SBB) operate the densest mixed railway traffic in the world. Due to the growing demand for mobility, SBB needs to increase the transportation capacity of the network by approximately 30% in the future. The "Flatland" Competition aims to address the vehicle rescheduling problem by providing a simplistic grid world environment[1].
Even if the challenge is open to any methodological approach, in this work the problem has been faced by means of reinforcement learning techniques: in a 2D grid environment representing the railway network, multiple agents with different goals must collaborate to maximize a global reward, reaching their target following the shortest possible path and avoiding deadlocks.

Chapter 2

Proposed Solution

The main problem of vehicle rescheduling is suitable for a Reinforcement Learning solution, where a large number of different environments are generated and trains (agents) are free to explore them and learn from the action they take, by means of reward functions, indicating positive and negative moves.
The considerable complexity of this task leads to start with a simpler scenario in which a single agent must reach his goal by following the shortest possible path, and then to consider a multi-agent scenario only after this problem is well-solved. The solution presented in this report is based on the *TreeObservation* proposed by the Flatland challenge with $tree\_depth = 2$ and on a shortest path predictor with $max\_depth = 20$, as can be visualized in Figure 2.1 The followed approach is aimed at improving the performance as the reward functions vary, with particular attention to the scalability of the network when trained on more or less complex environments.

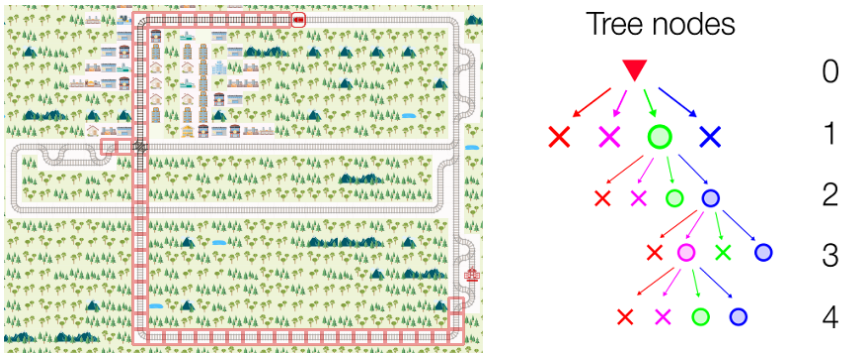

Figure 2.1: On the left, visual representation of the *TreeObservation* of an agent with $tree\_depth = 2$ and $max\_depth = 20$. On the right, Tree nodes of a generic railway network.

For the single agent case, only a penalty for each time step elapsed has been considered first, but clearly that's not representing the optimal solution. To improve the results, an important reward associated to the achievement of the destination has been added, as well as a reduced time-step penalty when the train gets closer to the target. Moreover, a standstill penalty has been evaluated, together with a lower decay rate for the epsilon variable, with the intention of encouraging a bit more the exploration of the map.
The model corresponding to the single agent best performance has been selected as the starting point for the multi-agent solution and evaluated. Some modifications on the reward function have been applied, mainly with the aim of strongly penalizing deadlocks, getting to an improvement in terms of trains trajectories rescheduling. After some useful tests, involving also a penalty for stopping on switches, the optimal reward function for the multi-agent setup has been chosen. Initially, a simple 3-agents scenario with a small map was considered, then the complexity of the task, which depends on the number of trains, available railways and environment size, has been gradually increased, up to 10 trains. In order to find the optimal tradeoff between number of deadlocks and number of agents reaching their station, many training processes have been performed, including some trials with "multi-level" training approach.
The proposed solution has been tested in both low and high complexity environments, initially considering only one possible speed and without malfunctions. The tests were then extended to the case with four different velocities and a non-zero malfunction rate.
All the files used for training and testing, plots and weights are inside the following GitHub repository [2].

Chapter 3

Network model

The Network architecture that has been chosen is the Dueling Double DQN (Deep Q-Network), which represents the state of the art of Value-based Reinforcement Learning techniques [3] [4]. The usage of this network has been deployed to solve a variety of games and problems, such as Doom Deathmatch [5].
In Flatland's framework, that has been considered, the Network does not include the three convolutional layers as the input of the neural network is not an image but a flattened vector, built starting from the observation. The implemented Network is shown in Figure 3.1.

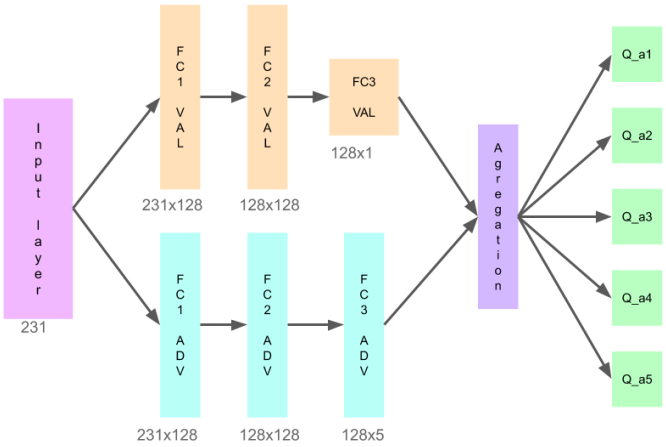

Figure 3.1: Dueling Double DQN applied to the Flatland's framework.

This choice reduces the complexity of the model as the useful information is already extracted and listed in the state. In fact the state, considering the *TreeObservation* with a maximum depth of 2 [6], results in a vector of 231 elements encoding information on the environment.

Chapter 4

Single Agent

The first problem that has been considered is the one with a single agent involved. The idea was to verify that the choice of the network and observations was good enough to achieve the goal and see if satisfactory results can be reached in this simplified framework.

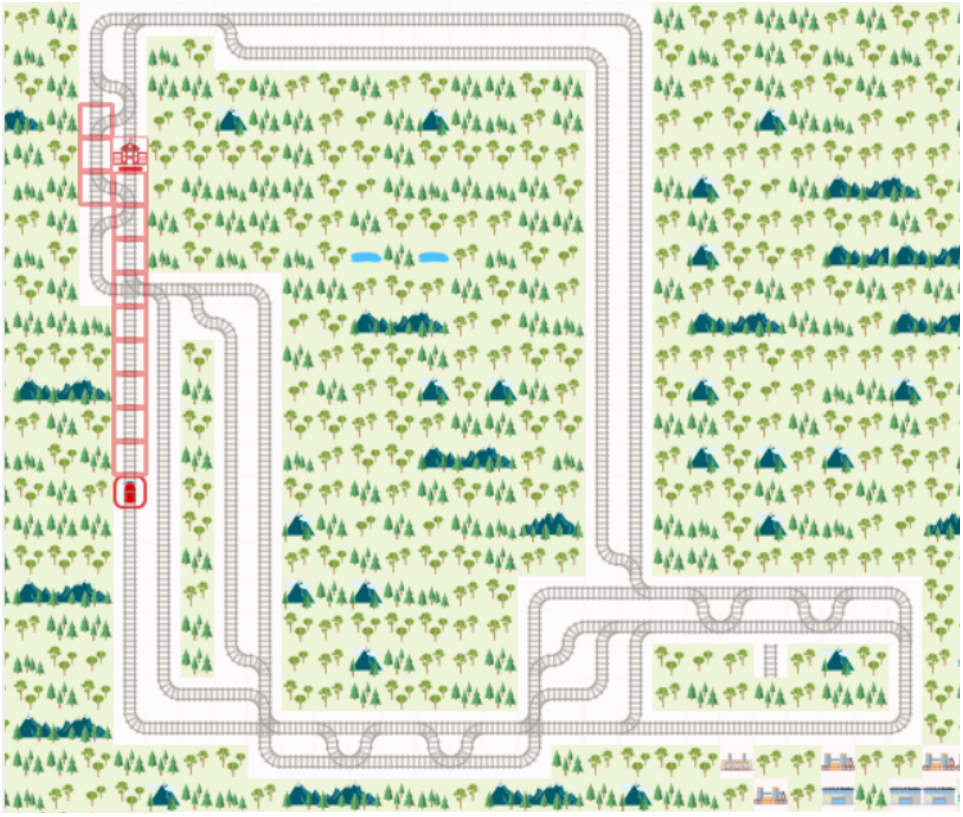

Figure 4.1: Single agent in a $35x35$ environment.

The first explorative training has been performed considering:

- $x\_dim = 35$
- $y\_dim = 35$
- $n\_agents = 1$
- $max\_num\_cities = 3$
- $max\_rails\_between\_cities = 2$
- $max\_rails\_in\_city = 3$

where x_dim and y_dim define the dimensions of the map and the last three values are parameters used by $sparse\_rail\_generator$, a railway generator provided by the organizers of the Challenge that mimics the real structure of a railway [6]. The velocity for the train is set to 1, this value is used by the $sparse\_schedule\_generator$ to set properly the agent's goal, [6]. The observation that has been considered is the $TreeObservationForRailEnv$, [6], with $max\_depth = 2$.
The default rewards consisting in:

- $+1$ when the goal is reached
- -1 given at each step to the agent

have been considered. The result of the training, considering 7000 iterations and a diminishing epsilon Figure 4.3 which is important for the exploration/exploitation trade-off [5], is shown in Figure 4.2.

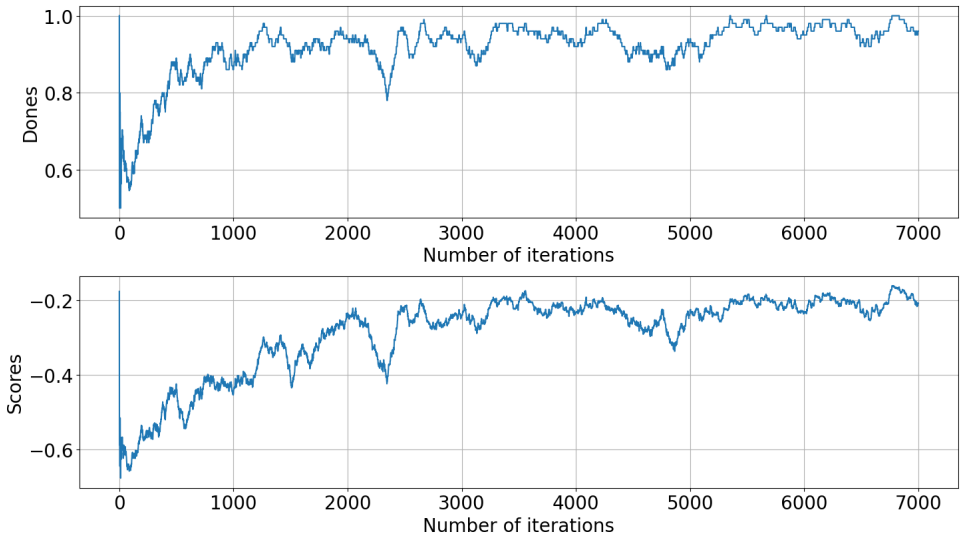

Figure 4.2: Dones and Scores metrics for the single agent first trial

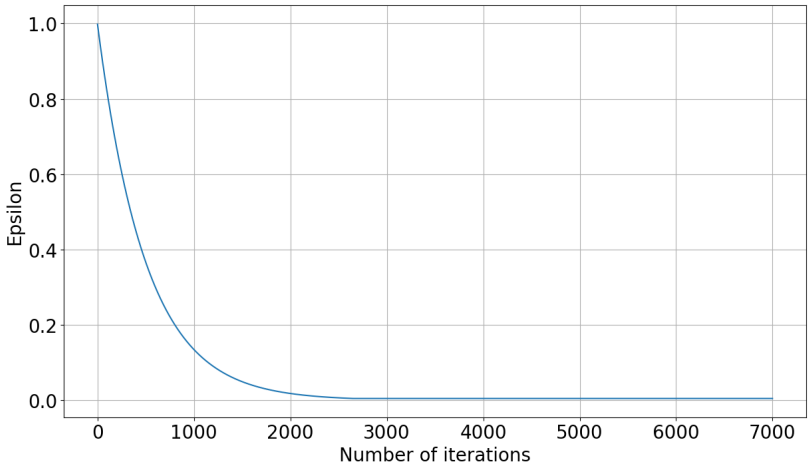

Figure 4.3: Decaying epsilon.

The result seemed promising as the 100% of dones has been achieved at some iterations. Results from the test-set can be seen in the Comparison Table (Fig. 4.6). Moreover, even with a local observation it has been possible to achieve good results, which confirms that at least in simple frameworks a global observation is not the only way to get results.
At that point, since there was still room for improvements also in this simplified framework, some modifications have been tested and will be presented in the next sections.

4.1 Reducing distance reward

The first intuition was that, even if the state contains information about the agent's goal, it can be interesting to modify the step penalty in order to penalize less the agent in case it gets closer to the target. The same idea was common also to the Netcetera's group which reached the 5th position in the 2019 Challenge [7]. The advantage in introducing this penalty should be to speed up the learning, improve dones and scores and force the agent to move. The choice for the rewards has been:

- $+10$ when the goal is reached
- -1 given generally at each step to the agent
- -0.5 given instead of the general one, in case the agent gets closer to the target

The choice of increasing the target reward, together with the choice of the reducing distance rewards, contributes to the result shown in Figure 4.4. Test-set outcomes can be seen in Fig. 4.6.

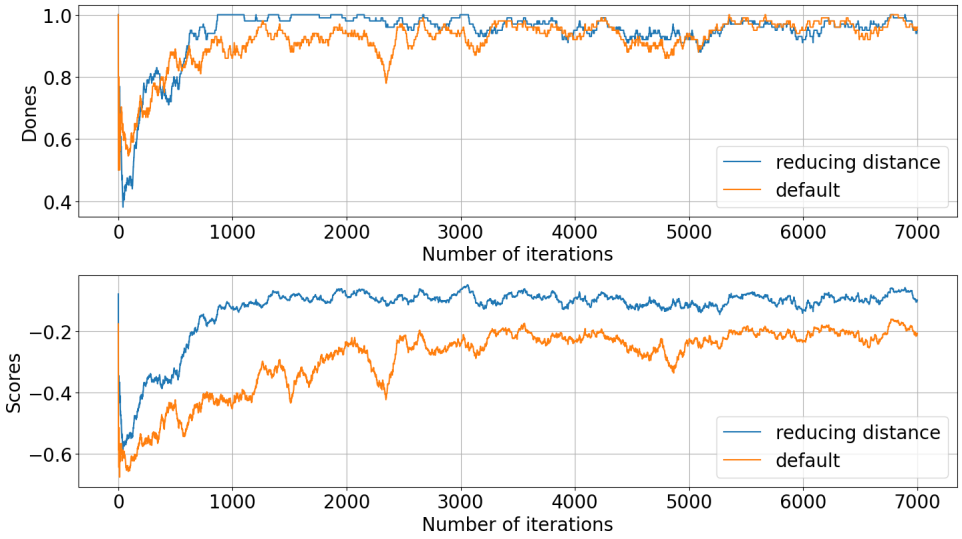

Figure 4.4: Dones and Scores for the single agent reducing distance.

The 100% of dones is reached sooner and it is maintained until the performance degrades over 4000 iterations and the result becomes similar to the default one. It is worth mentioning that the scores improves significantly but this is partially due to the fact that the reducing distance penalty is less than -1.
The choice of the epsilon at each iteration is the one shown in Figure 4.3, thus at less than 3000 iterations its value becomes 0.
In conclusion, in this single agent scenario the introduction of the reducing distance penalty does not degrades the performances in training but it improves them both in the dones plot and in the scores one.

4.2 Standstill penalty

Another intuition, coming from the observation of the behaviour of the agent trained using default parameters, is that there ins no reason why the train should be stopped at any time, since it will only be a cost and it wouldn't contribute in reaching the target.
Therefore it has been introduced a negative reward of -0.5 in case the agent does not perform any movement but it stays in standstill state. The result and comparison with the other results is shown in Figure 4.5.

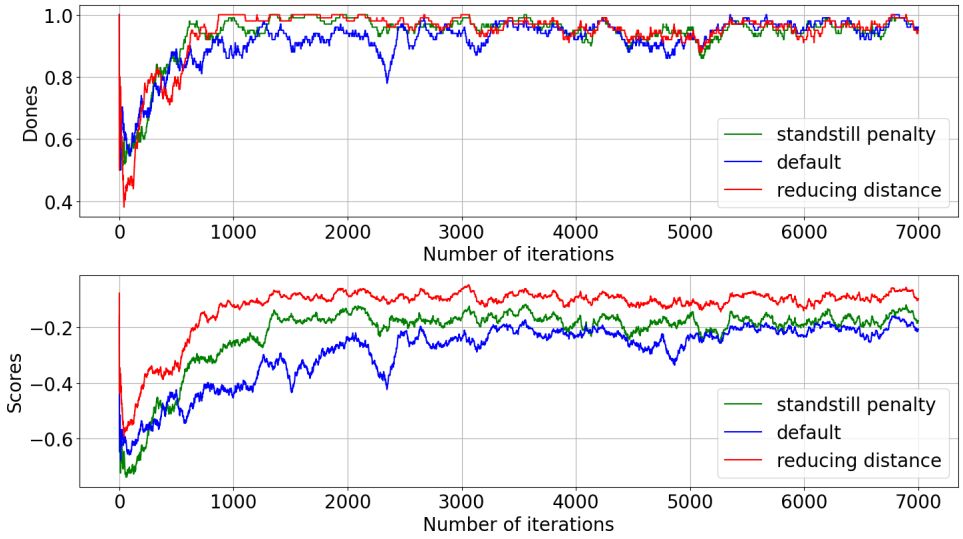

Figure 4.5: Dones and Scores for the single agent with standstill penalty.

The result is not so different with respect to the reducing distance one, this is reasonable as the lower penalty should teach the agent to move at each step in the target direction and not to stay standstill. However, it seems that this new reward brings less information with respect to the reducing distance one, having also worst performances both in the dones and scores metrics. Therefore, the reducing distance rewards should be preferred.
The results on the single agent seems satisfactory as almost 100% of Dones and a low value of scores has been achieved, moreover different possibilities has been explored and the chosen network model and observations gave positive feedback. Now, it is time to introduce higher complexities and move to the multi-agent scenario.

Metrics (avg over 1000 tests)	Default rewards	Reducing distance penalty	Standstill penalty
fraction of done	0,863	**0,951**	0,868
mean normalized return	-0,246	**-0,104**	-0,305

Figure 4.6: Comparison Table — Test set: 1 agent, 1000 iterations, 35x35 map

Chapter 5

Multi Agent

After reaching acceptable results with single agents, the training environment has been revisited to allow a 3-agents training, with the following parameters:

- $x\_dim = 40$
- $y\_dim = 40$
- $n\_agents = 3$
- $max\_num\_cities = 4$
- $max\_rails\_between\_cities = 2$
- $max\_rails\_in\_city = 3$

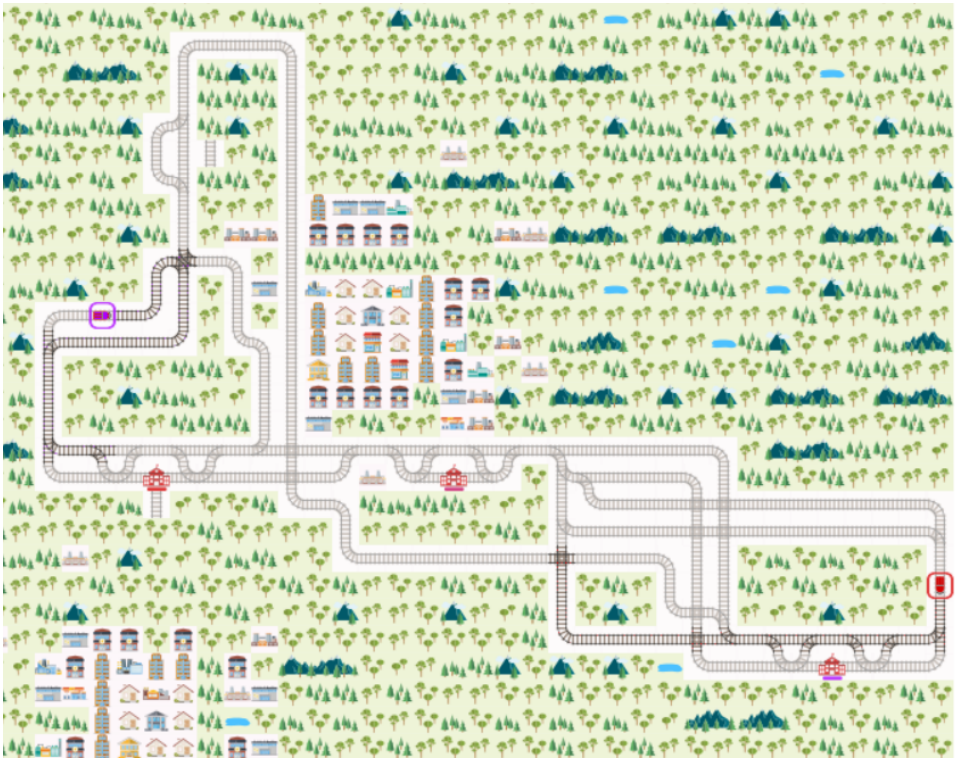

Figure 5.1: 3 agents in a $40x40$ environment.

First, it's important to point out that the observations used in all of the following trainings also include a *predictor*, more specifically the *ShortestPathPredictor* with a depth of 20. This object, included in the Flatland repository, is useful only in a multi-agent setup. Indeed it predicts the agents shortest paths to destination to allow the observer to detect possible future conflicts. This gives some useful information for the agents interactions.

5.1 Training 3 agents: first attempt

The first try was to start a training following the improvements learned with the single agents. Indeed two trainings (4000 iterations each) were compared, the first one setting the *reward* for reaching the goal to 10 and the latter also exploiting the *reduced_step_penalty* when agents' direction is towards the destination. The comparison in Figure 5.2 shows a counter-intuitive result with respect to what obtained with a single agent; indeed giving a reduced step penalty worsens the performance of 3 agents.

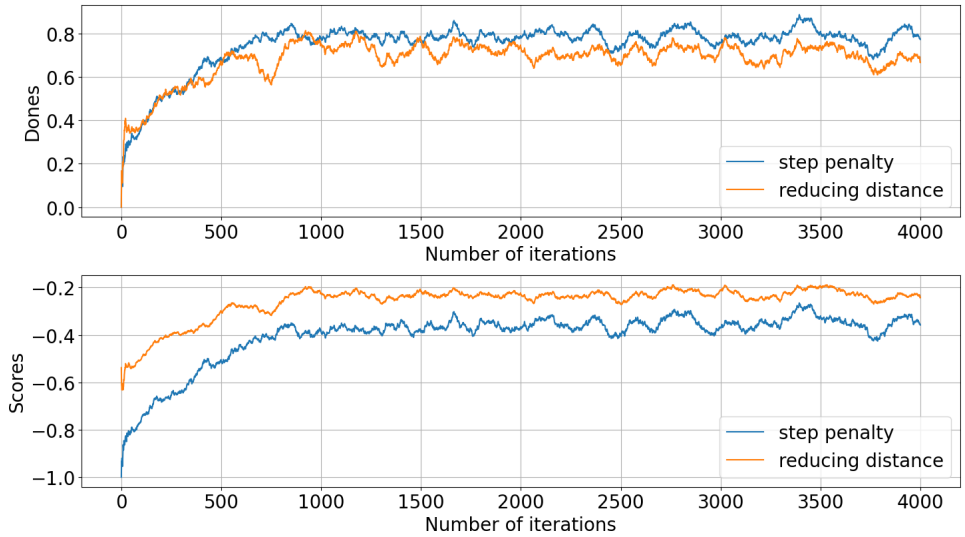

Figure 5.2: 3 agents training: fixed step penalty VS reducing distance penalty.

Even the best between these two trainings didn't give good results, indeed the tests (see Comparison table, Figure 5.8) reached respectively 79% and 70% of Dones. The main problem was in the interactions of the trains, which didn't seem to avoid each other, reaching deadlocks in several situations.

5.2 Training 3 agents: deadlock penalty implementation

In order to solve this issue, we decided to feed the agents with some information to understand whether they are in a deadlock. This info comes as a serious *penalty*, set to -10 and implemented into *rail_env.py* from the Flatland repository. As expected, results were very encouraging, reaching the following learning rate compared to the previous case (5.3). Agents learned to avoid each other by choosing alternative rails, and more surprisingly to sometimes stop before a switch to let others pass.

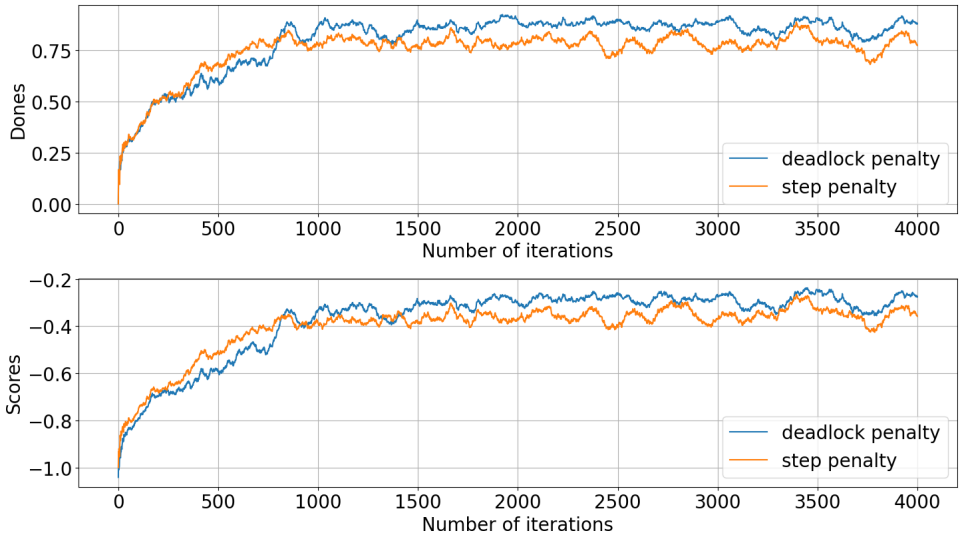

Figure 5.3: 3 agents training: deadlock penalty improvement.

In the test, we reached the following scores, shown in Table 5.8 (3 agents, Deadlock penalty).

5.3 Training 3 agents: stop on switch penalty implementation

The following trial to improve the results was to add a penalty for stopping on a switch [7], which is a critical situation that possibly prevents other agents from continuing their trajectory. Unfortunately, this attempt gave worse results than the previous training. The comparison can be seen in Figure 5.4, while the test results in the Table 5.8.

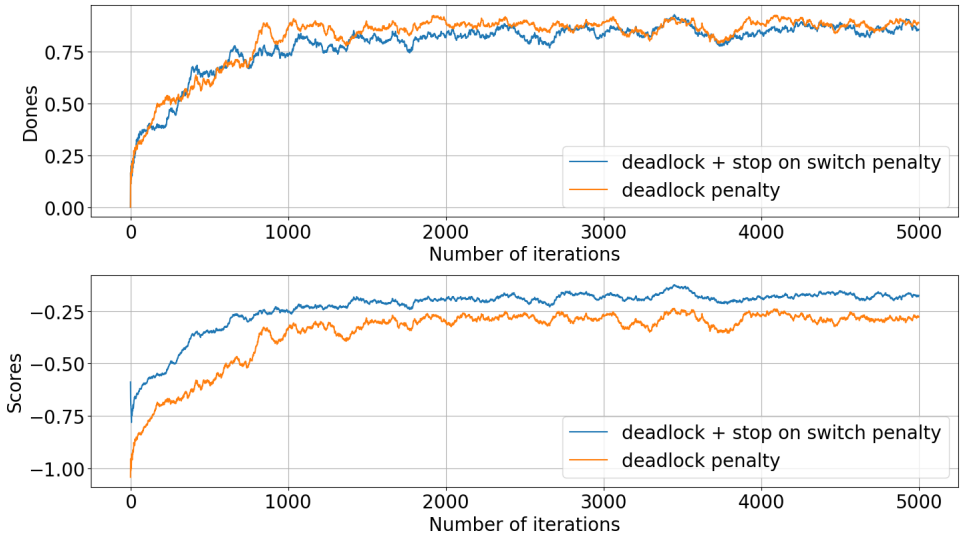

Figure 5.4: 3 agents training: stop on switch penalty.

5.4 Training 5 agents: first attempt

After achieving some improvements on agents' interactions, we moved to a slightly more difficult environment, concerning 5 agents on a bigger and more complex map. This possibly generates more complex situations to train on. The map has the following parameters:

- $x\_dim = 48$
- $y\_dim = 27$
- $n\_agents = 5$
- $max\_num\_cities = 5$
- $max\_rails\_between\_cities = 2$
- $max\_rails\_in\_city = 3$

As expected, the trained model performs way better than the 3 agents training. Test results are shown in Figure 5.8, while the learning curve in 5.5:

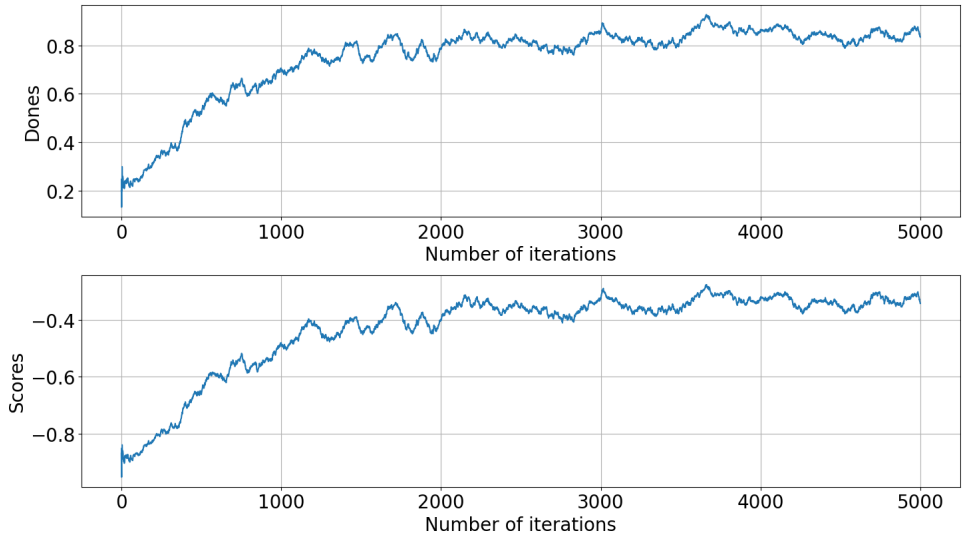

Figure 5.5: 5 agents training

5.5 Training 5 agents: mixed approaches

At this point, we wanted to explore some alternatives for improving the training, following the general rule of thumb for which great results can be achieved when training starts with a simple scenario and gets more difficult as iterations increase. For this reason, we started the training with a single agent in a big map (see *Training 10 agents : best result*) for 1000 *iterations*, and then on a medium size map (see *Training 5 agents : first attempt*) with 5 *agents* for 4000 iterations. This way, the single agents could optimize the path to destination and then multiple agents could learn to avoid deadlocks. As the single agent was performing better with a *reducing distance penalty*, we added it in this trial. Unfortunately, results were not better than the previous ones, as can be seen in Figure 5.6

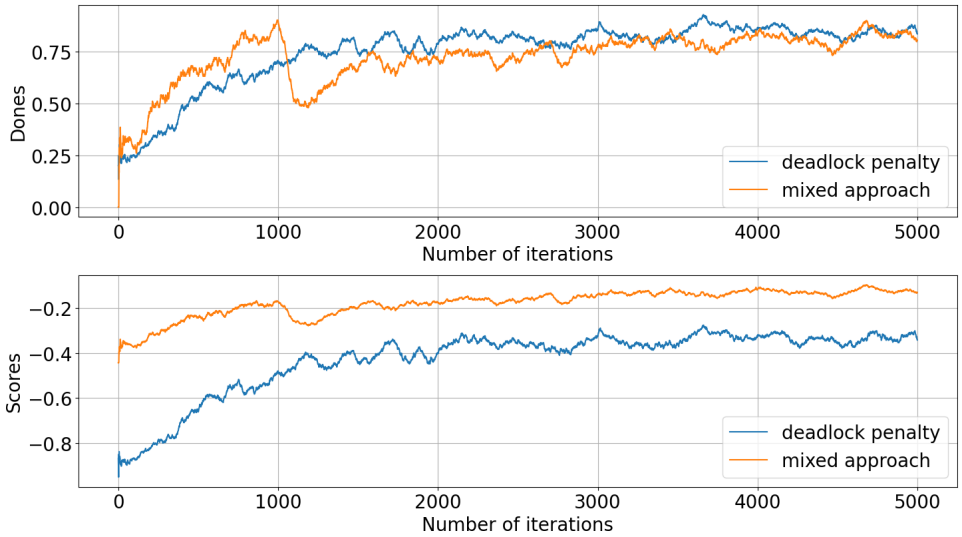

Figure 5.6: 5 agents training: previous approach VS mixed approach

Another approach, which will just be mentioned as it gave very bad results, was to train 5 agents on the two previously mentioned maps, which were alternated at every iteration.

5.6 Training 10 agents: best result

From what was experienced up to now, results have shown that training more agents on a bigger map is a good strategy, as long as there are enough agents on the map to ensure that the learning process focuses on their interactions. The last attempt was performed with these parameters:

- $x\_dim = 64$
- $y\_dim = 36$
- $max\_num\_cities = 5$
- $max\_rails\_between\_cities = 5$
- $max\_rails\_in\_city = 5$

7 agents on this size of the map were too few to ensure a good understanding of the deadlocks, so we repeated the computations with 10 agents, achieving the best performance of our study (Figure 5.7). Results from the Test set were also very encouraging (Table 5.8)

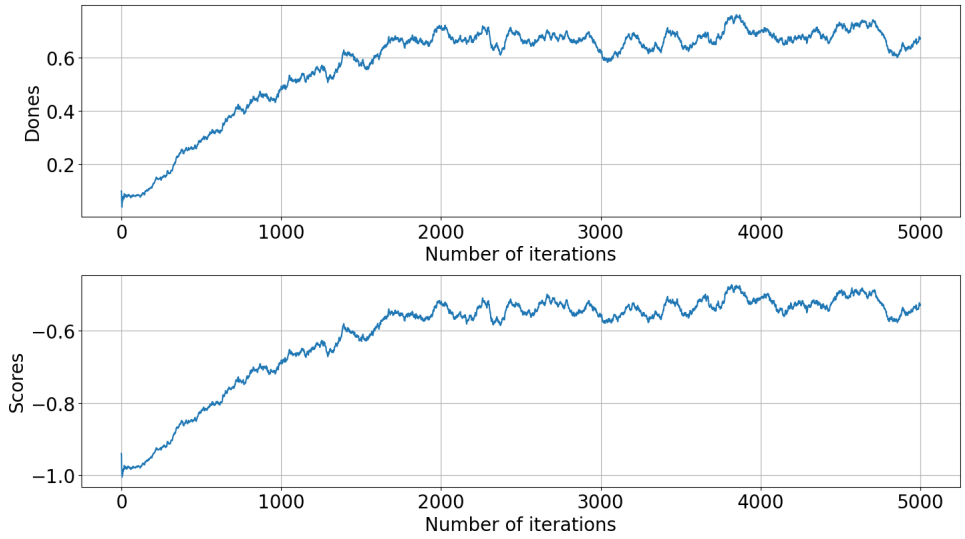

Figure 5.7: 10 agents training

	3 agents training			5 agents training		10 agents training
Metrics (avg over 1000 tests)	Fixed step penalty	Reducing distance penalty	Deadlock penalty	Deadlock penalty	Mixed Approach	Deadlock penalty
fraction of done-agents	0,792	0,709	0,875	0,887	0,865	**0,913**
fraction of deadlocks	\	\	0,0887	0,0133	0,0793	**0,0143**
mean normalized return	-0,339	-0,223	-0,275	-0,293	-0,285	**-0,238**

Figure 5.8: Comparison Table — Test set: 3 agents, 1000 iterations, 40x40 map.

5.7 Scalability

At this point of our study, it's important to point out an interesting trend. We experienced that training on a certain number of agents always gave better results when scaling down, during the tests, the complexity of the problem, i.e. reducing the map size or the number of the agent gives a better performance. The final results table gives some hint about this behavior. Scaling up the problem instead brings performance degradation.

5.8 Considerations about agents' action distribution

Last part of our study was done about the agents frequency of their actions. From figure (left, 5.9) we can notice that *LEFT* and *RIGHT*

actions are taken many times, and this is because Flatland's official framework considers these two actions as a $FORWARD$ without giving any invalid action penalty. For this reason, we modified *rail_env.py* to penalize this behavior. We see that there isn't a big performance degradation during the training (Figure 5.10), while now the $LEFT$ and $RIGHT$ actions are executed only when necessary (right, 5.9) . Anyway, we kept using the original $RailEnv$ in all other tests.

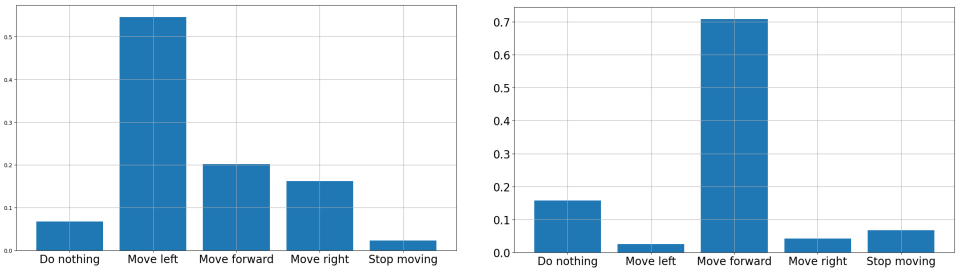

Figure 5.9: action distribution with the original RailEnv (left) and with the modified one (right)

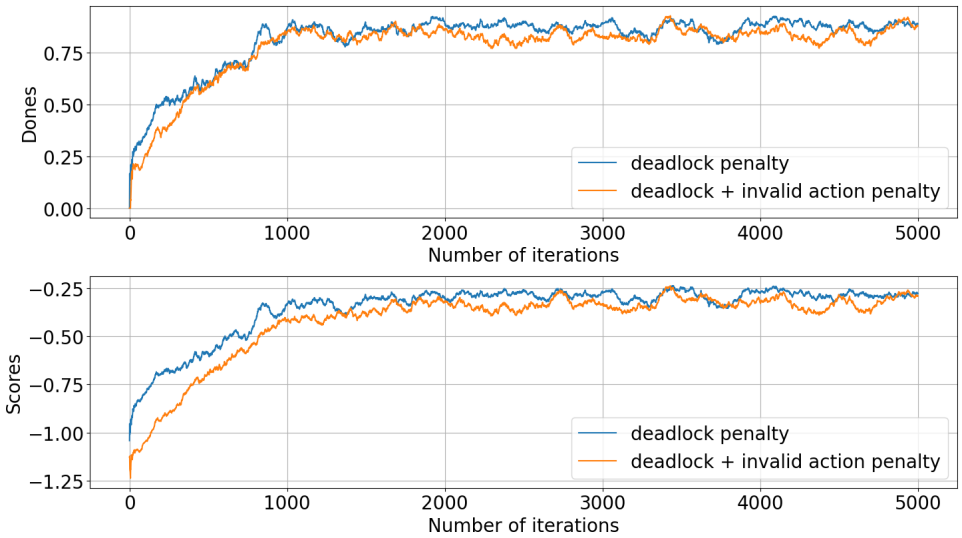

Figure 5.10: invalid action penalty

Chapter 6

Final Result

The results obtained by training the network with 10 agents on the "Big Size" environment were the best and are presented in this section. The applied rewards/penalties are:

- $step\_penalty$ = -1
- $global\_reward$ = 10
- $deadlock\_penalty$ = -10

The trained network has been tested on different sized maps and with different numbers of trains; in particular, two set of maps were selected, by mean of the official Flatland environment, specifically *RailEnv* from $flatland.envs.rail\_env.py$ and $sparse\_rail\_generator$ from $flatland.envs.rail\_generators$ [6]. These two test frameworks are described below, according to the parameters specified in the $rail\_env$ file:

- $rail\_env$ - Medium Size:
  - x_dim = 48
  - y_dim = 27
  - n_agents = 3 - 5 - 6
  - max_num_cities = 5
  - $max\_rails\_between\_cities$ = 2
  - $max\_rails\_in\_city$ = 3
  - $rand\_seed$ = 14

- $rail\_env$ - Big Size:

- $x\_dim = 64$
- $y\_dim = 36$
- $n\_agents = 5$ - 7 - 10
- $max\_num\_cities = 9$
- $max\_rails\_between\_cities = 5$
- $max\_rails\_in\_city = 5$
- $rand\_seed = 14$

Figure 6.1: Medium size test sample (left) and Big size (right)

500 iterations are considered for each test session.
Initially, tests were made on a scenario with no malfunctions and with all the agents going at the same speed (set to 1.0); the corresponding scores are collected in Figure 6.2.

	No malfunctions - single speed (1.0)					
	rail_env - Medium Size			rail_env - Big Size		
Metrics (avg over N tests)	3 agents	5 agents	7 agents	5 agents	7 agents	10 agents
fraction of done-agents	0.908667	0.8436	0.784	0.8236	0.787714	0.7016
fraction of deadlocks	0.015333	0.038	0.070286	0.0344	0.053143	0.1112
mean normalized return	-0.266862	-0.323886	-0.381059	-0.392504	-0.428224	-0.491795

Figure 6.2: Scores obtained in test phase with no malfunctions and single speed (1.0).

Then, to verify the generalization capabilities of the model, a more complicated scenario has been considered: four different speeds for the agents were introduced, all with 25% of probability, and also the possibility of malfunction for the agents has been added, according to the following parameters:

- $malfunction\_rate = 80$
- $min\_duration = 15$
- $max\_duration = 50$

The malfunction rate was very high, so that the model was tested on very harsh conditions. In this case, the maximum number of steps in an episode is tuned on the velocities, to allow slower train to reach their destination. The scores obtained during this second round of tests are collected in Figure 6.3.

	Malfunctions - 4 speeds (0.25 probability each)					
	rail_env - Medium Size			rail_env - Big Size		
Metrics (avg over N tests)	3 agents	5 agents	7 agents	5 agents	7 agents	10 agents
fraction of done-agents	0.808667	0.7308	0.752286	0.678	0.614857	0.6712
fraction of deadlocks	0.049333	0.0964	0.069429	0.082	0.134286	0.083
mean normalized return	-0.209396	-0.25544	-0.498329	-0.276325	-0.304061	-0.597517

Figure 6.3: Scores obtained in test phase with malfunctions and four different speeds.

Conclusions

After many comparisons, the best strategy turned out to be the one in which more agents are trained on a proportioned map, such that they could learn how to handle deadlocks, rescheduling their trajectories to optimally reach the targets. This solution offers better and better performances as the complexity of the environment decreases, i.e. scaling down the problem, by reducing the map size or the number of agents, obtaining a fraction of done-agents score always higher than 70%, while scaling up the problem brings performance degradation, but still satisfactory results.
The Netcetera group, which ranked fifth at the official Flatland challenge in January 2020, managed to get a 99% station arrival in the first round, and a 55% station arrival after the introduction of malfunctions and different speeds [7], within a sparse environment which is a bit larger but still comparable to our, where we got a score of 61% in the worst case, which results to be a good result. The winner of the competition reached 99% station arrival with malfunctions and different speeds [8].

References

[1] *Flatland Challenge*. URL: https://www.aicrowd.com/challenges/flatland-challenge.

[2] *GitHub repository of of the proposed solution*. URL: https://github.com/alessandrositta/Flatland_challenge.

[3] Hado V. Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty et al. Curran Associates, Inc., 2010, pp. 2613–2621. URL: http://papers.nips.cc/paper/3964-double-q-learning.pdf.

[4] Ziyu Wang, Nando de Freitas, and Marc Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581 (2015). arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581.

[5] *Deep Reinforcement Learning Course*. URL: https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/.

[6] *Flatland Challenge Documentation*. URL: http://flatland-rl-docs.s3-website.eu-central-1.amazonaws.com/.

[7] *Netcetera Blog*. URL: https://blog.netcetera.com/leverage-reinforcement-learning-for-building-intelligent-and-adaptive-trains-that-can-successfully-9f4cdef80162.

[8] *Flatland Challenge Leaderboards*. URL: https://www.aicrowd.com/challenges/flatland-challenge/leaderboards?.