Multi-Agent Path Finding via Tree LSTM

Yuhao Jiang*, Kunjie Zhang*, Qimai Li*, Jiaxin Chen†, Xiaolong Zhu
Parametrix.AI
Shenzhen, Guangdong, China
{yuhaojiang, ericzhang, qimaili, jiaxinchen, xiaolongzhu}@chaocanshu.ai

Abstract
In recent years, Multi-Agent Path Finding (MAPF) has attracted attention from the fields of both Operations Research (OR) and Reinforcement Learning (RL). However, in the 2021 Flatland3 Challenge, a competition on MAPF, the best RL method scored only 27.9, far less than the best OR method. This paper proposes a new RL solution to Flatland3 Challenge, which scores 125.3, several times higher than the best RL solution before. We creatively apply a novel network architecture, TreeLSTM, to MAPF in our solution. Together with several other RL techniques, including reward shaping, multiple-phase training, and centralized control, our solution is comparable to the top 2-3 OR methods.

1 Introduction
Multi-agent path finding (MAPF), i.e., finding collision-free paths for multiple agents on a graph, has been a long-standing combinatorial problem. On undirected graphs, a feasible solution can be found in polynomial time, but finding the fastest solution is NP-hard. And on directed graphs, even finding a feasible solution is NP-hard in general cases (Nebel 2020). Despite the great challenges of MAPF, it is of great social and economic value because many real-life scheduling and planning problems can be formulated as MAPF questions. Many operations research (OR) algorithms are proposed to efficiently find sub-optimal solutions (Cohen et al. 2019; Švancara et al. 2019; Ma et al. 2018; Ma, Kumar, and Koenig 2017).

As an important branch in decision theory, reinforcement learning has attracted a lot of attention these years due to its super-human performance in many complex games, such as AlphaGo (Silver et al. 2017), AlphaStar (Vinyals et al. 2019), and OpenAI Five (Berner et al. 2019). Inspired by the tremendous successes of RL on these complex games and decision scenarios, multi-agent reinforcement learning (MARL) is widely expected to work on MAPF problems as well. We study MARL in MAPF problems and aim to provide high-performance and scalable RL solutions. To develop and test our RL solution, we focus on a specific MAPF environment, Flatland.

Flatland (Mohanty et al. 2020; Laurent et al. 2021) is a train schedule simulator developed by the Swiss Federal Railway Company (SBB). It simulates trains and rail networks in the real world and serves as an excellent environment for testing different MAPF algorithms. Since 2019, SBB has successfully held three flatland challenges, attracting more than 200 teams worldwide, receiving thousands of submissions and over one million views. The key reasons why we focus on this environment are as follows,

- **Support Massive Agents**: On the maximum size of the map, up to hundreds of trains need to be planned.
- **Directed Graphs and Conflicts between Agents**: Trains CANNOT move back, and all decisions are not revocable. Deadlock occurs if the trains are not well planned (Figure 5), which makes this question very challenging.
- **Lack of High-Performance RL Solutions**: Existing RL solutions show a significant disadvantage compared with OR algorithms (27.9 vs. 141.0 scores).

To solve Flatland, we propose an RL solution by standard reinforcement learning algorithms at scale. The critical components of our RL solution are (1) the application of a TreeLSTM network architecture to process the tree-structured local observations of each agent, (2) the centralized control method to promote cooperation between agents, and (3) our optimized 20x faster feature parser.

Our contributions can be summarized as (1) We propose an RL solution consisting of domain-specific feature extraction and curriculum training phases design, a TreeLSTM network to process the structured observations, and a 20x faster feature parser to improve the sample efficiency. (2) Our observed strategies and performance show the potential of RL algorithms in MAPF problems. We find that standard RL methods coupled with domain-specific engineering can achieve comparable performance with OR algorithm (2nd-3rd OR). Our solution provides implementation insights to the MARL in the MAPF research community. (3) We open-sourced our solution and the optimized feature parser for further research on multi-agent reinforcement learning in MAPF problems.

---

*These authors contributed equally.
†Corresponding author

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

1https://github.com/liqimai/flatland-marl
2 Flatland3 Environment

Flatland is a simplified world of rail networks in which stations are connected by rails. Players control trains to run from one station to another. Its newest version, Flatland3, consists of the following rules.

- The time is discretized into timestamps from 0 to $T_{\text{max}}$.
- There are $N$ trains and several cities. Trains are numbered from 1 to $N$.
- Trains’ action space consists of five actions, \{do nothing, forward, stop, left, right\}. Trains are not allowed to move backward and must go along the rails.
- Trains have different speeds. Each train $i$ has its own speed $s_i$, and can move one step every $1/s_i$ turns. The time cost of one step $1/s_i$ is guaranteed to be an integer, and the largest possible speed is 1, i.e., one step a turn.
- For each train $i$, it has an earliest departure time $A_i$ and a latest arrival time $B_i$. Each train can depart from its initial station only after its earliest departure time $A_i$ and should try its best to arrive at its target station before the latest arrival time $B_i$.
- Trains randomly break down (malfunction) while running or waiting for departure. After the breakdown, the train must stay still for a period of time before moving again.

**Reward** The goal is to control all trains to reach target stations before their latest arrival time $B_i$. Every train will get a reward $R_i$ in the end. $T_i$ denotes the arrival time of each train.

- If a train arrives on time, then it scores 0.
- If it arrives late, it gets a negative reward $B_i - T_i$ as a penalty, according to how late it is.
- If a train does not manage to arrive at its target before the end time $T_{\text{max}}$, then the penalty consists of two parts, a temporal penalty and a spatial penalty. The temporal penalty is $B_i - T_{\text{max}}$, reflecting how late it is. The spatial penalty is decided by the shortest path distance $d_i$ between its final location at time $T_{\text{max}}$ and its target.

Formally, $R_i$ is defined as

$$ R_i = \begin{cases} 
0, & \text{if } T_i \leq A_i; \\
B_i - T_i, & \text{if } A_i < T_i \leq T_{\text{max}}; \\
B_i - T_{\text{max}} - d_i^{(T_{\text{max}})}, & \text{if } T_i > T_{\text{max}}; 
\end{cases} \quad (1) $$

where $d_i^{(T_{\text{max}})}$ is the distance between train $i$ and its target at time $T_{\text{max}}$.

$$ d_i^{(t)} = d \left( (x_i^{(t)}, y_i^{(t)}), \text{target}_i \right). \quad (2) $$

Our goal is to maximize the sum of individual rewards

$$ R = \sum_{i=0}^{N} R_i. \quad (3) $$

Apparently, $R$ is always non-positive, and $R = 0$ if and only if all trains reach targets on time. $|R|$ can be arbitrarily large, as long as the map size is sufficiently large and the algorithm performance is sufficiently bad.

**Normalized Reward** The magnitude of total reward $R$ greatly relies on the problem scale, such as the number of trains, the number of cities, the speeds of trains, and map size. To make rewards of different problem scales comparable, they are normalized as follows:

$$ \tilde{R} = 1 + \frac{R}{NT_{\text{max}}}, \quad (4) $$

where $N$ is the number of trains. The environment generating procedure guarantees $\tilde{R} \in [0, 1]$ by adjusting $T_{\text{max}}$. Normalized reward serves as the standard criterion for testing algorithms.

3 Our Approach

The RL solution we provide is cooperative multi-agent reinforcement learning. Each agent independently observes a part of the map localized around itself and encodes the part
Table 1: Extracted agent attributes in $X^{\text{attr}}$.

| Name                        | Dim. | Type |
|-----------------------------|------|------|
| Train ID                    | 1 int|      |
| Earliest departure time     | 1 int|      |
| Latest arrival time         | 1 int|      |
| Initial direction           | 4 one-hot| |
| Initial distance to target  | 1 int|      |
| Road type                   | 11 one-hot| |
| Possible transitions        | 16 binary| |
| Current direction           | 4 one-hot| |
| Last-turn direction         | 4 one-hot| |
| Deadlocked or not           | 1 binary| |
| Valid actions               | 5 binary| |
| Distance to target          | 1 int|      |
| Current time                | 1 int|      |
| #turns before late          | 1 int|      |
| Arrival time                | 1 int|      |
| State                       | 7 one-hot| |
| Is off-map state            | 1 binary| |
| Is on-map state             | 1 binary| |
| Is malfunction state        | 1 binary| |
| Is moving or not            | 1 binary| |
| Malfunction ends            | 1 binary| |
| Left malfunction turns      | 1 int|      |
| Speed state-machine         | 5 int|      |

Table 2: Useful information extracted for each node in spanning tree.

| Description                                | Dim. | Type |
|--------------------------------------------|------|------|
| #agents in same direction                  | 1 int|      |
| #agents in opposite direction              | 1 int|      |
| #agents ready to depart                    | 1 int|      |
| #agents in malfunction                     | 1 int|      |
| Distance to agent’s own target             | 1 int|      |
| Distance to other agents’ targets, if any  | 1 int|      |
| Distance to other agents, if any           | 1 int|      |
| Distance to potential conflict, if any     | 1 int|      |
| Distance to unusable switch, if any        | 1 int|      |
| Branch length                              | 1 int|      |
| Slowest speed of observed agents           | 1 float| |

Figure 3: Construct a spanning tree for each agent to encode its possible future paths. Each node represents a branch the agent may choose.

3.1 Feature Extraction
The extracted features consist of two parts, $X^{\text{attr}}$ and $X^{\text{tree}}$.

Agent Attributes The first part, $X^{\text{attr}} = \{x_{i}^{\text{attr}}\}_{i=1}^{N}$, are attributes of each agent, such as ID, earliest departure time, latest arrival time, their current state, direction, and the time left, etc. See Table 1 for detailed contents of $X^{\text{attr}}$.

Tree Representation of Possible Future Paths The second part $X^{\text{tree}}$ is the main part of the observations. It encodes possible future paths of each agent as well as useful information about these paths into a tree-like structure. We take the rail network as a directed graph and construct a spanning tree for each agent by a depth-limited BFS (breadth-first-search) starting from its current location. Each node in the tree represents a branch the agent may choose, see Figure 3. Formally, the spanning tree we construct for each train $i$ is $T_{i} = (V_{i}, \mathcal{E}_{i})$ with node set $V_{i}$ and edge set $\mathcal{E}_{i}$. Each node $\nu \in V_{i}$ is associated with a vector $x_{\nu}$, containing useful information about this branch. See Table 2 for detailed contents of $x_{\nu}$. So, $X^{\text{tree}}$ contains both the tree structure and these associated node features:

$$X^{\text{tree}} = \{(T_{i}, X^{\text{tree}}_{i})\}_{i=1}^{N},$$

where $X^{\text{tree}}_{i} = \{x_{\nu} | \nu \in V_{i}\}$.

Such tree representation is provided by Flatland environment and has also been explored by other RL methods (Mohanty et al. 2020; Laurent et al. 2021). However, no RL method before has achieved comparable performance as ours because we made the following improvements.

- First, all methods before concatenate node features together into a long vector so that it can be fed into MLPs. Normal networks can only process vectors, not tree-like input. After concatenation, the underlying structures of trees are lost. In contrast, we think the tree structures are super important for decision-making and must be preserved, as they encode map topology. In section 3.2, we processing such tree-structured data by a special neural network structure, TreeLSTM (Tai, Socher, and Manning 2015).

- Second, our trees are much deeper than others. Tree depth decides the range of agents’ field of view and thus
significantly affects the performance. Extracting tree representation is a computationally intensive task. The flatland3 built-in implementation of tree representation is super slow because of the poor efficiency of Python language and the unnecessary complete ternary tree it uses, so the RL methods before have a very limited tree depth, typically 3. We re-implement tree construction by C++ and prune complete ternary trees into normal trees. Our implementation is 20x faster than the built-in one and enables us to build trees with depths of more than 10.

- Third, we build trees in the BFS manner, while the built-in implementation is in the DFS manner. Constructing a spanning tree in a DFS manner makes some nodes near the root on the graph become far from the root, which is a disadvantage.

### 3.2 Neural Network Architecture

As shown in Figure 4, our neural networks first process \( X^{\text{attr}} \) by a 4-layer MLP and process \( X^{\text{tree}} \) by TreeLSTM (Tai, Socher, and Manning 2015). TreeLSTM is a variant of LSTM designed for tree-structured data, whose details will be elaborated later.

\[
H^{\text{attr}} = \text{MLP}(X^{\text{attr}}) \tag{6}
\]

\[
H^{\text{tree}} = \text{TreeLSTM}(X^{\text{tree}}) \tag{7}
\]

Then, we concatenate \( H^{\text{attr}} \) and \( H^{\text{tree}} \) together and feed them into three consecutive self-attention blocks to encourage communications between agents. With the self-attention mechanism (Vaswani et al. 2017), a train is able to be aware of other trains’ observations and forms its own knowledge of the global map.

\[
H^{(0)} = [H^{\text{attr}}, H^{\text{tree}}], \tag{8}
\]

\[
H^{(l)} = \text{Self-Attention}\left(H^{(l-1)}\right), \quad l = 1, 2, 3. \tag{9}
\]

Finally, \( H^{(3)} \) is fed into two different heads to obtain final actions logits \( A \in \mathbb{R}^{N \times 1} \) and estimated state-value \( v \in \mathbb{R} \).

\[
A = \text{MLP}(H^{(3)}) \tag{10}
\]

\[
V = \text{MLP}(H^{(3)}) \tag{11}
\]

\[
v = \sum_{i=0}^{N} V_i \tag{12}
\]

**TreeLSTM**  
LSTM (Hochreiter and Schmidhuber 1997), as a kind of RNN, was designed to deal with sequential data. Each LSTM cell takes state \((c_{t-1}, h_{t-1})\) of last cell and a new \( x_t \) as input, and output new cell state \( c \) and \( h \) to next cell.

\[
(c_t, h_t) = \text{LSTM-Cell}(x_t, (c_{t-1}, h_{t-1})) \tag{13}
\]

Sequential data is a special case of trees, where each node has a unique child — its successor. Tai, Socher, and Manning modified its structure to deal with general trees in 2015. Tree differs from sequential data in the allowed number of children. Unlike sequential data, nodes in a tree are allowed to have multiple children. As a result, TreeLSTM receives a set of children’s output as input instead:

\[
(c_t, h_t) = \text{TreeLSTM-Cell}(x_t, S_t), \tag{14}
\]

where \( S_t = \{(h_k, c_k) \mid k \in \text{Child}(t)\} \). Within TreeLSTM cells, there are several ways to aggregate children’s states, leading to different variants of TreeLSTM. In our network, we adopt Child-sum TreeLSTM. See Tai, Socher, and Manning (2015) for detailed structures of TreeLSTM cells.

### 3.3 Reward Design

Agents are given rewards at every time step, according to their performance within the moment. Besides the normalized reward generated by the environment, agents are also rewarded when they depart from stations, arrive at targets, and get penalized when deadlocks happen. To promote cooperation between them, these rewards are shared by all agents, and no credit assignment is performed. As a result, a single agent is encouraged to wait for others if the waiting can lead to global efficiency improvement.

![Figure 4: Overview of our network architecture. FC stands for fully connected layer.](image-url)
Environmental Reward Agents are rewarded environmental reward $r_t^{(e)}$ at time step $t$:

$$r_t^{(e)} = \bar{R}_t,$$  
(15)

where $\bar{R}_t$ is the normalized environmental reward agents get in time step $t$.

Departure Reward We reward agents when there are new agents departing:

$$r_t^{(d)} = \frac{n_t^{(d)} - n_{t-1}^{(d)}}{N},$$  
(16)

where $n_t^{(d)}$ is the number of agents departing at or before time step $t$.

Arrival Reward We reward agents when there are new arrivals:

$$r_t^{(a)} = \frac{n_t^{(a)} - n_{t-1}^{(a)}}{N},$$  
(17)

where $n_t^{(a)}$ is the number of arrival so far at time step $t$.

Deadlock Penalty Because trains are not allowed to go backward, if two trains go into a single rail in opposite directions, a deadlock happens (see Figure 5). So, we give a penalty when new deadlocks happen:

$$r_t^{(l)} = \frac{n_t^{(l)} - n_{t-1}^{(l)}}{N},$$  
(18)

where $n_t^{(l)}$ is the number of deadlocks on the map at time step $t$.

Total Reward The final reward we give to agents at time $t$ is a weighted sum of all terms above:

$$r_t = c_r r_t^{(e)} + c_a r_t^{(a)} + c_d r_t^{(d)} - c_l r_t^{(l)},$$  
(19)

where $c_r, c_a, c_d, c_l$ are weight parameters.

---

**Figure 5:** Because trains are not allowed to go backward, if two trains go into a single rail in opposite directions, a deadlock happens.

---

**Environmental Reward** Agents are rewarded environmental reward $r_t^{(e)}$ at time step $t$:

$$r_t^{(e)} = \bar{R}_t,$$  
(15)

where $\bar{R}_t$ is the normalized environmental reward agents get in time step $t$.

**Departure Reward** We reward agents when there are new agents departing:

$$r_t^{(d)} = \frac{n_t^{(d)} - n_{t-1}^{(d)}}{N},$$  
(16)

where $n_t^{(d)}$ is the number of agents departing at or before time step $t$.

**Arrival Reward** We reward agents when there are new arrivals:

$$r_t^{(a)} = \frac{n_t^{(a)} - n_{t-1}^{(a)}}{N},$$  
(17)

where $n_t^{(a)}$ is the number of arrival so far at time step $t$.

**Deadlock Penalty** Because trains are not allowed to go backward, if two trains go into a single rail in opposite directions, a deadlock happens and no train can pass this rail again (see Figure 5). So, we give a penalty when new deadlocks happen:

$$r_t^{(l)} = \frac{n_t^{(l)} - n_{t-1}^{(l)}}{N},$$  
(18)

where $n_t^{(l)}$ is the number of deadlocks on the map at time step $t$.

**Total Reward** The final reward we give to agents at time $t$ is a weighted sum of all terms above:

$$r_t = c_r r_t^{(e)} + c_a r_t^{(a)} + c_d r_t^{(d)} - c_l r_t^{(l)},$$  
(19)

where $c_r, c_a, c_d, c_l$ are weight parameters.

---

**Table 3:** Settings of different phases.

| Model | Phase-I | Phase-II | Phase-III-50 | Phase-III-80 |
|-------|---------|----------|--------------|--------------|
| #agents | 50 | 50 | 50 | 80 |
| Reward Weights | $c_r$ | $c_a$ | $c_d$ | $c_l$ |
| Initialized by | N/A | Phase-I | Phase-II | Phase-III-50 |

**Table 4:** Compare Phase-I and Phase-II.

| Test Stage | Phase-I | Phase-II | Phase-III-50 | Phase-III-80 |
|------------|---------|----------|--------------|--------------|
| Arrival% | 70.0 | 86.2 | 84.9 | 92.9 |
| Env. Reward | 0.859 | 0.920 | 0.933 | 0.957 |
| Depart% | 79.0 | 99.2 | 89.9 | 97.2 |

---

**4 Experiments**

**4.1 Experiment Settings**

We largely followed the final round (round 2) configurations of the Flatland3 challenge to conduct experiments so that our results are comparable with the ones on the challenge leaderboard. There are 15 test stages in the final round, and each stage contains 10 test cases. Problem scales (Table 6) and difficulty gradually increase from initial stages to advanced stages. The first stage is the smallest one with 7 agents on a $30 \times 30$ map, while the last stage contains 425 agents on a $158 \times 158$ map. Teams’ submissions are tested stage by stage. A team can proceed to the next stage only if they pass the last stage (arrival ratio reaches 25%).

**4.2 Multiple Phase Training**

To reduce training difficulty, we train our models in curriculum learning style, and the whole training process can be roughly divided into three phases. In phases I and II, we train a model in 50-agent environments. We found that the learned model generalizes well to smaller environments but not larger ones. In phase III, models are initialized by the

---

https://www.aicrowd.com/challenges/flatland-3/leaderboards
one learned in phase II and fine-tuned in settings with more agents.

**Phase I** Initially, we only use the environmental reward, arrival reward, and deadlock penalty to encourage the trains to march on their targets and avoid deadlocks (see Phase-I in Table 3). After training, 70% agents in 50-agent environments can reach their targets, and a normalized reward of 0.859 is achieved. However, 21% agents have never departed because of the deadlock penalty. Agents choose not to depart and behave conservatively to avoid being penalized by deadlocks.

**Phase II** In phase I, many trains do not depart, so we add a departure reward to encourage more departures (see Phase-II in Table 3). The experiment is initialized by the phase-I model, and after 5 days of training, its arrival ratio increases to 86.2%, and almost all the trains have departed (Table 4).

**Phase III** Finally, we deal with environments with more than 80 agents. Training models in such large environments from scratch is very difficult, so we adopt curriculum learning. Models for large environments are initialized by parameters learned in small environments. Although models learned in small environments are able to directly generalize to large environments (Table 5), fine-tuning in large environments increases performances significantly.

### 4.3 Results and Analysis

Our stage-specific results are reported in Table 6; Final scores as well as top 10 teams’ scores in Flatland3 challenge are listed in Table 7. In summary, we scored 125.3, ranking top 2–3 on the leaderboard, while the best RL method before scored only 27.9. More specifically, we observe the following phenomena:

- No RL method before us managed to pass Test_03 stage, while our method passed all 15 stages.
- While the number of agents increases, model performance decreases, which suggests large-scale problems are more difficult than we expected.
- When the numbers of agents are equal (Test_04 to Test_08), model performance increases with a larger map

| Test Stage | Model       | #agents | Map Size | #cities | Arrival% | Env. Reward | Avg. Time/s |
|------------|-------------|---------|----------|---------|----------|-------------|-------------|
| Test.00    | Phase-III-50| 7       | 30 × 30  | 2       | 94.3 ± 10.0 | .957 ± .030 | 7.022       |
| Test.01    | Phase-III-50| 10      | 30 × 30  | 2       | 92.0 ± 9.2  | .947 ± .047 | 8.430       |
| Test.02    | Phase-III-50| 20      | 30 × 30  | 3       | 87.0 ± 13.6 | .934 ± .063 | 16.486      |
| Test.03    | Phase-III-50| 50      | 30 × 35  | 3       | 86.2 ± 10.2 | .922 ± .047 | 32.292      |
| Test.04    | Phase-III-80| 80      | 35 × 30  | 5       | 62.6 ± 11.9 | .812 ± .051 | 40.580      |
| Test.05    | Phase-III-80| 80      | 45 × 35  | 7       | 62.9 ± 12.8 | .824 ± .049 | 60.009      |
| Test.06    | Phase-III-80| 80      | 40 × 60  | 9       | 70.6 ± 6.2  | .859 ± .028 | 99.566      |
| Test.07    | Phase-III-80| 80      | 60 × 40  | 13      | 65.4 ± 12.6 | .833 ± .051 | 109.386     |
| Test.08    | Phase-III-80| 80      | 60 × 60  | 17      | 74.3 ± 9.6  | .877 ± .029 | 160.928     |
| Test.09    | Phase-III-100| 100     | 80 × 120 | 21      | 59.7 ± 15.7 | .795 ± .067 | 480.971     |
| Test.10    | Phase-III-100| 100     | 100 × 80 | 25      | 57.6 ± 16.7 | .779 ± .067 | 346.861     |
| Test.11    | Phase-III-200| 200     | 100 × 100| 29      | 52.8 ± 5.8  | .790 ± .033 | 488.549     |
| Test.12    | Phase-III-200| 200     | 150 × 150| 33      | 57.3 ± 5.0  | .777 ± .037 | 1314.509    |
| Test.13    | Phase-III-200| 400     | 150 × 150| 37      | 34.9 ± 7.2  | .704 ± .031 | 2029.058    |
| Test.14    | Phase-III-200| 425     | 158 × 158| 41      | 39.3 ± 9.7  | .721 ± .038 | 2329.925    |

Table 6: Performance of our models in 15 test stages.

| Rank | Team               | Tag     | Score | Arv.% |
|------|--------------------|---------|-------|-------|
| 1    | An_OldDriver       | OR      | 141.0 | 88.0  |
| 2    | Zain               | OR      | 132.5 | 88.8  |
| -    | Ours               | RL      | 125.3 | 66.4  |
| 3    | SmartTrains        | OR      | 118.0 | 76.9  |
| 4    | dsa                | OR      | 107.5 | 44.2  |
| 5    | stavros_kakoulidis | other   | 104.5 | 55.6  |
| 6    | UniTeam            | other   | 29.9  | 39.1  |
| 7    | WaveTeam           | RL      | 27.9  | 38.6  |
| 8    | SOBA               | RL      | 27.8  | 32.6  |
| 9    | ChewChewChew       | RL      | 20.0  | 30.6  |
| 10   | fridayPhenom       | OR      | 6.9   | 18.6  |

Table 7: Compare our results with the top 10 in Flatland3 challenge leaderboard. The scores are collected by accumulating environmental rewards from all 15 test stages.

| Test Stage | #agents | Arrival% | Env. Reward |
|------------|---------|----------|-------------|
| Test.03    | 50      | 56.4 ± 10.8 | .815 ± .038 |
| Test.04    | 80      | 50.7 ± 9.6  | .810 ± .023 |
| Test.05    | 80      | 47.4 ± 9.0  | .794 ± .021 |
| Test.06    | 80      | 53.0 ± 8.0  | .815 ± .041 |
| Test.07    | 80      | 59.1 ± 7.9  | .805 ± .041 |
| Test.08    | 80      | 60.4 ± 6.0  | .812 ± .021 |
| Test.09    | 100     | 56.7 ± 10.3 | .802 ± .035 |
| Test.10    | 100     | 54.7 ± 3.9  | .801 ± .031 |

Table 8: Performance of model Phase-III-200 in small environments. Generally, it performs worse than the scale-specific models in Table 6.
and more cities because it leads to lower agent density and less traffic congestion.

- Compared to the third-best team, we achieved a higher environmental reward but a lower arrival ratio. This indicates that environmental rewards are not always consistent with arrival ratios because arrival ratios only care agents arrive or not while environmental rewards also care about how fast agents arrive. They are two highly related but different objectives. Similar phenomena can be observed in team An_Old_Driver and team Zain.

**Generalization across environment scales** We are also interested in the generalization ability of our models and particularly interested in the generalization across environment scales. As Table 5 shows, models learned in small environments are able to generalize to large environments but perform worse than the one fine-tuned in large environments. Table 8 shows that models specialized in large environments are also able to generalize to small environments but perform worse than the ones learned in small environments. To achieve optimal performance, we need to train multiple scale-specific models.

**Agent Cooperation** We observed many self-organized cooperative patterns in agents’ behaviors. They learn to line up to march in a compact manner (Figure 7). Fast ones learned to overtake the slow ones (Figure 6). Slow trains make way for fast ones (Figure 9). When there are two parallel rail lanes, trains spontaneously line up as if they are in a two-way street (Figure 8).

### 5 Conclusion

We provided a new RL solution to the Flatland3 challenge and achieved a score 4x better than the best RL method before. The key reasons behind the improvement are 1) the tree features and TreeLSTM we adopt and 2) the 20x faster feature parser, which enables us to train our model with far more data than the RL methods before. However, there is still a gap between our method and state-of-the-art OR methods (Li et al. 2021). Our method also takes longer time than OR methods. Another drawback is that there lacks a single model that is able to handle environments of any scale. To achieve optimal performance, we have to train multiple scale-specific models.

### References

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Cohen, L.; Uras, T.; Kumar, T. S.; and Koenig, S. 2019. Optimal and bounded-suboptimal multi-agent motion planning. In *Twelfth Annual Symposium on Combinatorial Search*.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Laurent, F.; Schneider, M.; Scheller, C.; Watson, J.; Li, J.; Chen, Z.; Zheng, Y.; Chan, S.-H.; Makhnev, K.; Svidchenko, O.; Egorov, V.; Ivanov, D.; Shpilman, A.; Spirovská, E.; Tanevski, O.; Nikov, A.; Grunder, R.; Galevski, D.; Mitrovski, J.; Sartoretti, G.; Luo, Z.; Damani, M.; Bhattacharya, N.; Agarwal, S.; Egli, A.; Nygren, E.; and Mohanty, S. 2021. Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, 275–301.

Li, J.; Chen, Z.; Zheng, Y.; Chan, S.-H.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2021. Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 477–485.

Ma, H.; Kumar, T. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Ma, H.; Wagner, G.; Felnner, A.; Li, J.; Kumar, T.; and Koenig, S. 2018. Multi-agent path finding with deadlines. *arXiv preprint arXiv:1806.04216*.

Mohanty, S.; Nygren, E.; Laurent, F.; Schneider, M.; Scheller, C.; Bhattacharya, N.; Watson, J.; Egli, A.; Eichenberger, C.; Baumberger, C.; et al. 2020. Flatland-RL: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893*.

Nebel, B. 2020. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 212–216.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *ArXiv*, abs/1707.06347.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354–359.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 7732–7739.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1556–1566.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Vinyls, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.
Figure 6: Fast trains overtake the slow ones. The speed of the blue train is 0.25 while the other three are 1.0. Fast ones overtake the slow ones to reach targets earlier.

Figure 7: Trains line up in a row.

Figure 8: When there are two parallel rail lanes, trains spontaneously line up as if they are in a two-way street.

Figure 9: A slow train makes way for a fast one. The orange one has a speed of 1.0, and the blue one has a speed of 0.5. The blue one waits until the orange one passes.