Scalable Rail Planning & Replanning

Winning the 2020 Flatland Challenge

Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan,
Daniel Harabor, Peter J.Stuckey, Hang Ma, Sven Koenig

Presented by: LYMENG SIM

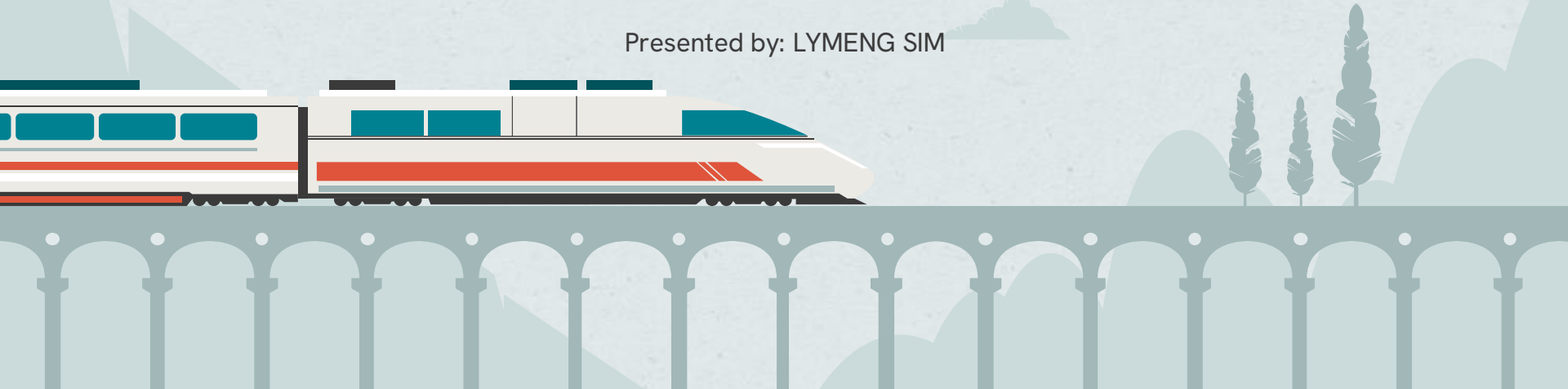
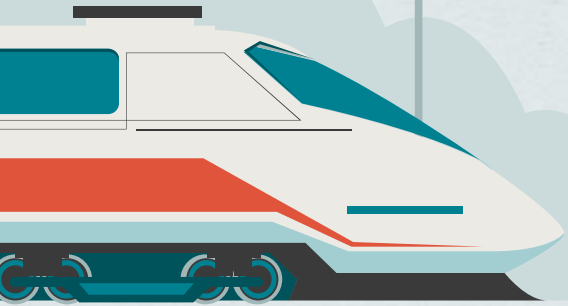


Table of contents

01

Introduction



AIcrowd

Introduction

Problem

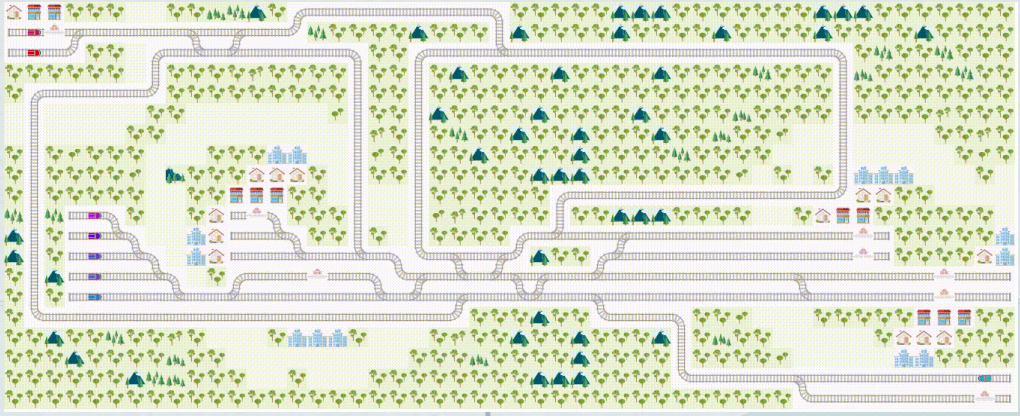
"How to efficiently manage dense traffic on complex rail networks?"

Goal

To design a plan so that each train moves from its start station to its target station within a give time limit.

MAPF

finding collision-free paths for multiple agents to move from their starting positions to their goal locations in a shared, grid-based environment.



700
participants

51
countries

2,000
submissions

64
teams in Round 1

44
teams in Round 2

An_Old_Driver

Jiaoyang Li

University of Southern California, USA

Zhe Chen

Monash University, Australia

Yi Zheng

University of Southern California, USA

Shao-Hung Chan

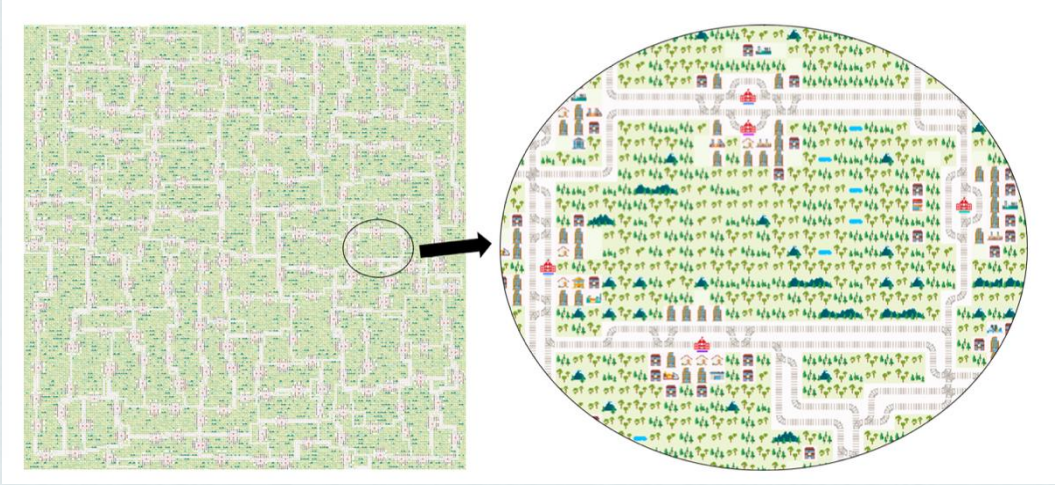
University of Southern California, USA

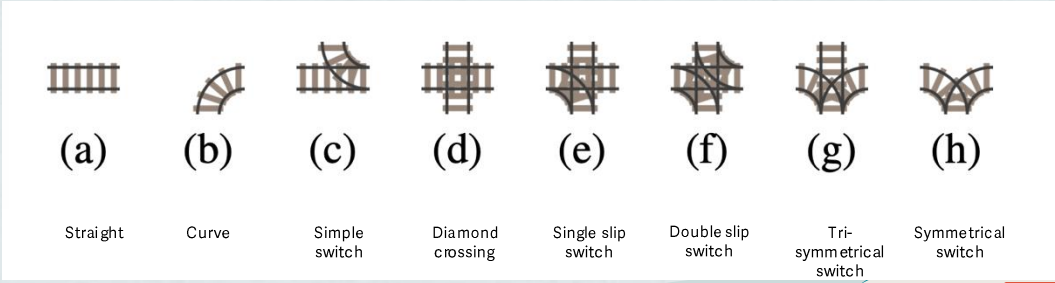
02

Flatland Environment

Environment

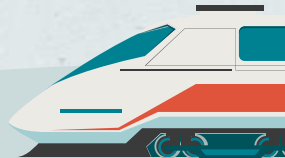
- the competition was built on the flatland framework that provide random generated environment for the agents to operate in. each environment contained a w x h rectangular grid of cells and each cells contain “Rail”.
- there are 8 type of rails in this competition.





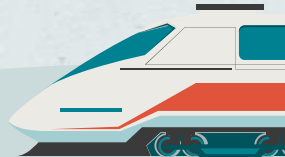
Agents

- **Agents** were assigned a starting cell, a direction {N, E, S, W} available at the origin, and a target cell where agents need to reach.
- The starting cells are always different from target cells. Both starting cells and target cells are Rail Type (a) and could be reached from both direction.
- Every agents issued 1 of 5 actions (MOVE_FORWARD), (MOVE_LEFT), (MOVE_RIGHT), (STOP) = halt and (DO_NOTHING) = no_op, but **never** go back or go in reverse.
- if 2 agents "**collided**" while heading in opposite directions, they become deadlocked for the rest of episode and could not reach the targets.
- An agent could also **malfunction**, in that case the agents' actions has no effect and could not move until the malfunction was resolved.



Task

- the Task was to bring as many agents to their targets in as little timesteps as possible.
- penalty of -1 for each timestep they did not attain their target, or vice versa.
- additional reward of 1 if all the agents had reached their targets.
- The participants have to solve as many environment as possible within 8hour time limit. The overall score S with *m* being number of environment solved.



2.2 Relationship with MAPF and Its Variants

1. Movement Constraint

Train can only move along predefined rail tracks and cannot move backward, while MAPF, agent can move freely in all directions.

2. Dynamic entry and exit

Trains enter the environment over time and leave it after reaching their target cells, which is related to online MAPF, where all agents are typically present from the beginning. (Svancara et al. 2019).

3. Not all agents must succeed

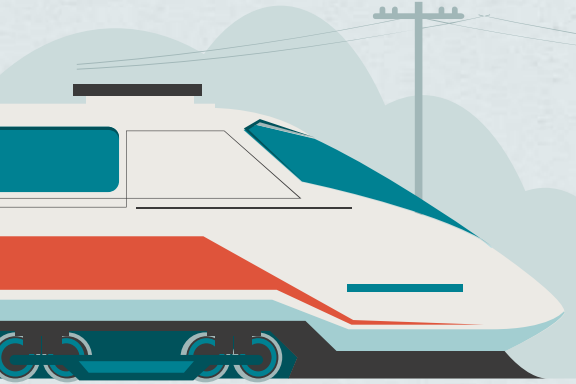
the objective is slightly different. Instead of requiring all agents to reach their targets, the goal is to move as many trains as possible within a limited time.

4. Uncertainty

Trains break down randomly while moving and remain then stationary at the breakdown location for a number of timesteps, which is related to MAPF while delay probabilities of stochastic travel time.

03

Round 1



400

instances

14

settings

8 hours

including planning and simulation time

test	m	w	h	n	λ	#instances
Test_0	5	25	25	≤ 2	$\leq 1/50$	50
Test_1	10	30	30	≤ 2	$\leq 1/100$	50
Test_2	20	30	30	≤ 3	$\leq 1/200$	50
Test_3	50	20	35	≤ 3	$\leq 1/500$	40
Test_4	80	35	20	≤ 5	$\leq 1/800$	30
Test_5	80	35	35	≤ 5	$\leq 1/800$	30
Test_6	80	40	60	≤ 9	$\leq 1/800$	30
Test_7	80	60	40	≤ 13	$\leq 1/800$	30
Test_8	80	60	60	≤ 17	$\leq 1/800$	20
Test_9	100	80	120	≤ 21	$\leq 1/1000$	20
Test_10	100	100	80	≤ 25	$\leq 1/1000$	20
Test_11	200	100	100	≤ 29	$\leq 1/2000$	10
Test_12	200	150	150	≤ 33	$\leq 1/2000$	10
Test_13	400	150	150	≤ 37	$\leq 1/4000$	10

Table 1: The settings of the 400 instances used in Round 1.

3.1 Finding the Approach to Path Compatibility

Conflict Based Search (CBS)

- Use space-time A* to to find collision-free paths for all agents while minimizing total travel time
- CBS first plan paths for all agents independently and then detect conflicts between them.
- When a conflict is found, the algorithm adds constraints to avoid that conflict and replans the paths.
- Continues until all conflicts are resolved, resulting in a collision-free solution.

Why CBS Fails ?

- Many trains starting from the same location, and many trains sharing narrow tracks
- CBS resolves conflicts mostly in a pairwise manner, but in this environment, conflicts often involve many agents at once, especially in crowded areas.
- the algorithm needs to explore a very large number of possibilities, which makes it extremely slow.

Optimality

scalability
+
efficiency

Prioritized Planning

Algorithm

Trains are planned one by one, based on a priority order.

Drawback

Prioritized Planning does not guarantee optimal solutions, and the quality of the solution depends heavily on the chosen priority order.

Tried but Fail

Group PP: grouping trains and planning them together using CBS.

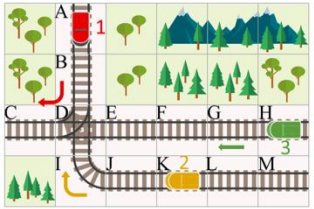


Figure 3: Three-train example.

Train 1: no constraints

Train 2: avoid train 1

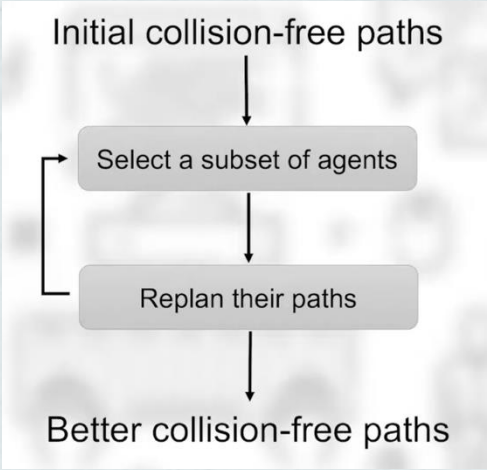
Train 3: avoid train 2

Large Neighborhood Search (LNS)

Algorithm

Select a small group ➡ Remove their paths ➡
Replan only these train ➡ Evaluate the new solution ➡
Accept or Reject ➡ Repeat ↩

- **train-based strategy:** select a train *a* with the largest delay and trains that prevent train *a* from reaching its target cell earlier;
- **the intersection-based strategy:** select trains that visit the same intersection.



Parallel LNS

To further improve performance, the authors run multiple LNS processes in parallel.
Each process uses a different priority ordering, and at the end, the best solution is selected.

1. in order of the train indices: lowest to highest,
2. in order of the earliest arrival time: highest to lowest,
3. in order of the earliest arrival time: lowest to highest, and
4. in order of trains, with smallest earliest arrival time, which first group based on starting locations, then choose the best from each group.
   e.g. 3 Stations: A, B, C.- each has 2 trains
   the order will be: A1, B1, C1, A2, B2, C2 not: A1, A2, B1, B2, C1, C2

- LNS reduce the flowtime - on 312 out of 400 over PP with average reduction of 12.4%
- Parallel LNS reduced the flowtime on 157 instance over LNS, with average reduction 11.1%

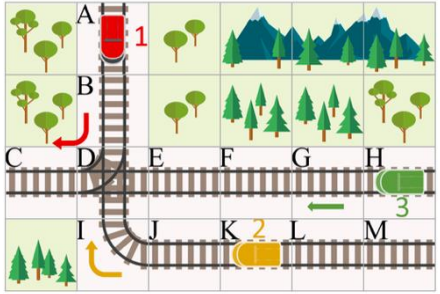
3.2 Recovering from Breakdowns

Minimum Communication Policies (MCP)

MCP avoids such deadlocks by stopping some trains to maintain the ordering with which each train visits each cell.

Partial Replanning

- When a disruption occurs, the system identifies the affected trains.
- It removes their current paths.
- Then, it replans new paths for these trains, while keeping all other trains fixed.



Path for train 1: [A, B, D, C]

Path for train 2: [K, J, I, D, B, A]

Path for train 3: [H, G, F, E, D, C]

3.3 Fighting with Simulator

Asymmetric Movement Rule

In this environment, trains cannot freely move in all directions. Once a train moves forward, it cannot simply reverse its direction.

Command Commitment

Once a train starts a move, you cannot change your decision until it finishes that move.

Round 1 Summary

PP

To generate an initial solution

LNS

To improve its quality

MCP

To generate action command

Partial Planning

To improve the solution quality when a train encounters a new malfunction

Scoring (mean reward) of **-0.104** with a success rate **100%**

3% and **10%** higher than 2nd and 3rd place

Round 2

Round 2 Requirement

- Require maximizing the accumulated reward over an infinite number of instance of increasing difficulty within 8 hours
- The instances were grouped into tests of increasing difficulty. Each test contains 10 levels with different malfunction rates, where Level 0 does not have malfunctions ($\lambda = 0$), and Level j ($1 \leq j \leq 9$) has **malfunction rate $\lambda = 1/250j$**

4.1 Fighting with simulator again

Symmetric Movement Rule

- Remove dependency on train order
- All trains move simultaneously
- Can enter a cell vacated at same timestep
- Collisions defined by space-time only

Simulation Runtime

- Simulator runs every timestep
- Slow even when idle
- High cost for large instance

👉 “70% time in simulator, 30% in algorithm”

4.2 Trade off between Runtime and Reward

Problem

- Limited total time (8 hours)

Trade off

- More LNS : better quality, fewer instances
- Less LNS : more instances, lower quality

Solution

- Simulated annealing
- Parallel LNS with Leader Thread
- Partial Replanning

Simulated Annealing

- Use SA to choose the number of iterations for which we should run LNS on the instances in each test to allocate the runtime spend by LNS.

Experiment

- first run experiments to measure how LNS affects runtime and reward.
- then build a model to estimate the effect of different numbers of LNS iterations.
- Using this model, they apply simulated annealing to allocate LNS iterations across different test instances (ranging best from 5 - 1230).
- Finally, they show that this strategy improves total reward, even though it slightly reduces the number of solved instances.
- Solved 1 fewer instance within 8h and improve accumulated reward by 0.709

Parallel LNS with Leader Thread

Algorithms

- The authors improve LNS by running multiple versions in parallel.
- Use a leader thread based on the best priority ordering as a safe baseline.
- At the same time, run additional threads that explore alternative solutions.
- If any thread finds a better solution, it is accepted. Otherwise, the leader's result is used.

Result

✓ Parallel LNS with Leader thread consumed **9s** fewer and improve the accumulated reward by 0.571 among instance in Test_2 to Test_27.

Partial Replanning

Algorithms

- For large instance, PP is time consuming since many trains are malfunctional and many affected by malfunctioning trains
- PP is less effective for large instances in term of reward sine they have large value trains and arrival times.
- Apply PP for instance in **Test_1** (2 trains) to **Test_28** (631 trains) only.

Result

- PP consumed 1,965 seconds and improved their accumulated reward by 4.812.
- Eventually, solved 2 fewer instances within 8 hours but improved the accumulated reward by 3.629

4.3 Speed is all that matters

For large instances with many trains, the number of timesteps is usually large, and thus the search space of space-time A* is also large, which leads to unacceptable runtimes.

Safe Interval Path Finding

- SIPP is an improved version of Space-Time A* that reduces the search space by grouping safe timesteps into intervals.
- **Space-Time A* :** Check: t=1,2,3,4,5...
- **SIPP :** Check: [0–2], [4–∞]

Result: solved 6 more instances and improved the accumulated reward by 4.576.

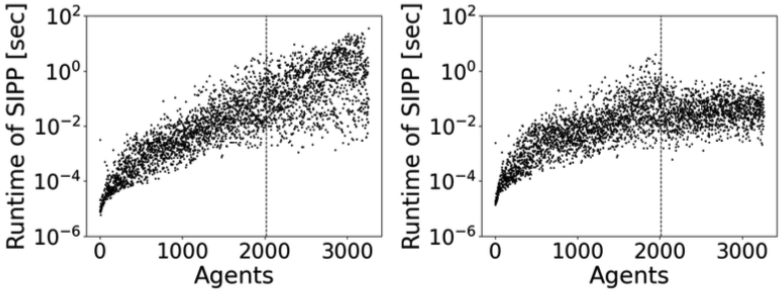
4.3 Speed is all that matters

Lazy Planning

- only a subset of trains is planned initially, and the remaining trains are planned during execution.
- has 2 benefits for large instances with non-zero malfunction rates:
  1. can prevent severe traffic jams
  2. the new paths can result in smaller arrival times eventually.
- **Result**: solve 5 more instances, improved the accumulated reward by 3.282.

With SIPP

- Test_36 Level_0 with 3256 trains
- 10^{-5}s to 34s for a single path finding



(a) Without lazy planning (b) With lazy planning

Round 2 Summary

PP + SIPP

To generate an initial solution

Parallel LNS

With leader thread to improve its quality
SA to determined LNS iterations

MCP & Partial Planning

To generate action command and to improve the solution quality when new malfunction occurs

Lazy Planning

To continue planning path during execution

Solved **362** instances with success rate **98.5%** within 8h

and score **297.507, 1.16%** and **24.168%** higher than 2nd and 3rd place

Thanks!

Do you have any questions?