# Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge

**Jiaoyang Li,[1] Zhe Chen,[2] Yi Zheng,[1] Shao-Hung Chan,[1]**
**Daniel Harabor,[2] Peter J. Stuckey,[2] Hang Ma,[3] Sven Koenig[1]**

[1]University of Southern California, USA
[2]Monash University, Australia
[3]Simon Fraser University, Canada
{jiaoyanl, yzheng63, shaohung, skoenig}@usc.edu, {zhe.chen, daniel.harabor, peter.stuckey}@monash.edu, hangma@sfu.ca

## Abstract

Multi-Agent Path Finding (MAPF) is the combinatorial problem of finding collision-free paths for multiple agents on a graph. This abstract describes MAPF-based software for solving train planning and replanning problems on large-scale rail networks under uncertainty. The software recently won the 2020 Flatland Challenge, a NeurIPS competition trying to efficiently manage dense traffic on rail networks. The software incorporates many state-of-the-art MAPF or, in general, optimization technologies, such as prioritized planning, safe interval path planning, parallel computing, simulated annealing, large neighborhood search, and minimum communication policies. It can plan collision-free paths for thousands of trains within a few minutes and deliver deadlock-free actions in real-time during execution.

## Introduction

The 2020 Flatland Challenge is a NeurIPS competition set up to answer the question "How to efficiently manage dense traffic on complex rail networks?" It consists of an idealized rail planning problem. Given a map showing rail tracks and train stations (see Figure 1) and a set of $m$ trains with start and target stations, we need to move the trains from their start stations to their target stations so that no two trains occupy the same track segment (a vertex collision) or cross each other by moving in opposite directions from adjacent track segments (an edge collision) simultaneously. Trains may malfunction during execution. That is, a train may stop at a random timestep for a random duration. Our goal is to maximize the number of trains that reach their target stations before the given deadline and minimize their arrival times.

The competition was supported by three large European railway network operators and involved more than 700 participants from 51 countries making more than 2,000 submissions over 4 months. We outperformed all other entries in both rounds, including all Reinforcement Learning entries. In this abstract, we focus on the framework of our software for the main round, which asked us to maximize the accumulated reward over an infinite number of instances of increasing difficulty within 8 hours. Please see our technical paper (Li et al. 2021b) and video demonstration at https://youtu.be/Pw4GBL1UhPA for more details.



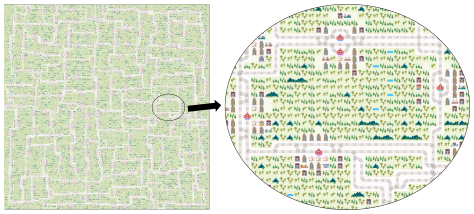

Figure 1: Flatland environment represented by a $229 \times 229$ grid map. The grey lines represent rail tracks, and the red houses on the rail tracks represent train stations.

## System Framework

The academic version of the competition is called Multi-Agent Path Finding (MAPF), which is moving multiple agents from their start vertices to their target vertices on a graph while avoiding vertex and edge collisions. Our software (https://github.com/Jiaoyang-Li/Flatland) incorporates many state-of-the-art MAPF or, in general, optimization technologies. Figure 2 shows the high-level framework.

### Offline Planning Phase (Before Timestep 0)

In the planning phase (with a runtime limit of 10 minutes), we take as input the map and the start and target stations of the trains and plan collision-free paths (called a MAPF plan) for them under the assumption that no malfunctions occur.

**Prioritized Planning (PP).** We use PP, a popular MAPF algorithm, to generate an initial MAPF plan. PP first sorts all trains according to a priority ordering and then, from the highest-priority train to the lowest-priority train, plans a shortest path for each train that does not collide with any already planned path. We use SIPP (Phillips and Likhachev 2011), an advanced version of A*, to plan each path.

**Large Neighborhood Search (LNS).** Although PP can find a MAPF plan rapidly, its quality is sometimes far from optimal. We thus follow Li et al. (2021a) by repeating a neighborhood search process via LNS to improve the quality of the MAPF plan until an iteration limit is reached. In each iteration, we select a subset of trains and replan their paths using PP. The new paths need to avoid collisions with each other and with the paths of the other trains.

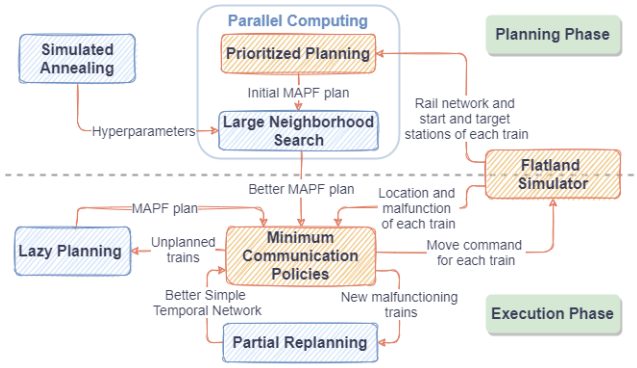


Figure 2: System framework. The modules for generating a feasible solution are colored in yellow while the modules for improving the solution quality are colored in blue.

**Simulated Annealing (SA).** As the competition imposes an overall runtime limit of 8 hours with an infinite number of instances, we need to trade-off solution quality with runtime. We thus collect training data offline and use SA to determine the LNS iteration limits for instances of different sizes.

**Parallel Computing.** As the competition provides 4 CPUs for evaluation, we run in parallel 4 PPs with different priority orderings followed by 4 LNSes and pick the best plan.

### Online Execution Phase (After Timestep 0)

In the execution phase (with a runtime limit of 10 seconds per timestep), we handle malfunctions and deliver deadlock-free move commands to the trains at each timestep.

**Minimum Communication Policies (MCP).** When trains malfunction during execution, deadlocks could happen if all trains stick to their original paths. MCP (Ma, Kumar, and Koenig 2017) avoids the deadlocks by using simple temporal networks to stop some trains so as to maintain the ordering in which trains visit each rail segment.

**Partial Replanning (PR).** MCP sometimes stops trains unnecessarily. We developed a PR mechanism to avoid such unnecessary waits. When train A encounters a malfunction at some timestep, we collect all switches and crossing rail segments that train A is going to visit in the future and then collect the trains that are going to visit at least one of these switches or crossing rail segments after train A. We use PP to find better paths for these trains one at a time.

**Lazy Planing (LP).** The single-agent path planner is slow when there are thousands of trains to schedule because, as the paths of more trains are planned, it has to plan paths that avoid collisions with an increasing number of existing paths, resulting in its runtime growing rapidly. We thus used an LP scheme that plans paths for only some of the trains during the planning phase, then lets the trains move, and plans paths for the rest of the trains during the execution phase.

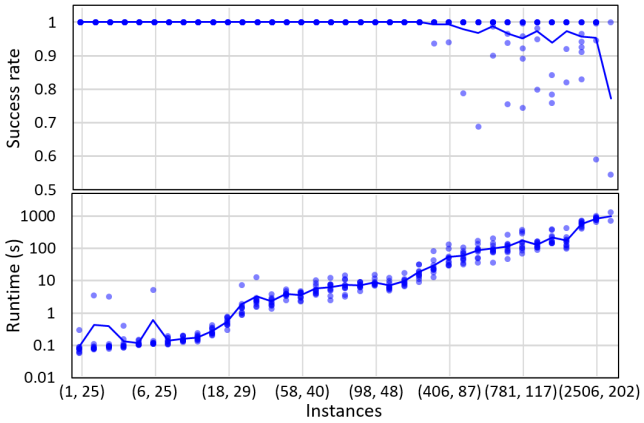


Figure 3: Success rate (= percentage of trains completed before the given deadline) and runtime of the system on the leaderboard. $x$-axis labeled $(m, n)$ represents instances with $m$ trains on $n \times n$ grids. Each instance size has 10 dots (= 10 instances), with the lines indicting their mean values.

LP has two benefits: (1) It avoids pushing too many trains into the environment at once, which can prevent severe traffic congestion, and (2) it takes into account the influence of the malfunctions that have already happened.

## Performance

Figure 3 shows the performance of our system on the leaderboard. The runtime of our software was probably much smaller than what is shown in the figure as the runtime shown in the figure also includes the runtime of the Flatland simulator, which consumed 70% of the 8-hour runtime budget when we ran the evaluation on our local server.

## Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, and 1935712 as well as a gift from Amazon. The Research at Monash University was supported by the Australian Research Council (ARC) under grant numbers DP190100013 and DP200100025 as well as a gift from Amazon. The research at Simon Fraser University was supported by the Natural Sciences and Engineering Research Council (NSERC) under grant number RGPIN-2020-06540.

## References

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021a. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *IJCAI*, (in print).

Li, J.; Chen, Z.; Zheng, Y.; Chen, S.-H.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2021b. Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge. In *ICAPS*, 477–485.

Ma, H.; Kumar, T. K. S.; and Koenig, S. 2017. Multi-Agent Path Finding with Delay Probabilities. In *AAAI*, 3605–3612.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe Interval Path Planning for Dynamic Environments. In *ICRA*, 5628–5635.