Flatland challenge team report

JelDor team

10.01.20

Although the challenge was positioned as being a machine learning one (with the emphasis on reinforcement learning), our team decided to choose heuristic search as the main tool to solve the problem. We believe that state-of-the-art heuristic search algorithms and accompanying techniques are perfectly suitable for solving multi-train path finding/scheduling problems.

Our algorithm is essentially a planning one, i.e. paths (one per each agent) are planned and then agents just follow them when simulation starts. When some agent is delayed re-planning is invoked.

We rely on prioritized planning as it is known to be one of the fastest (in terms on computation time) types of planning compared to other multi-agent path finding (MAPF) algorithms. When agents start off the map and disappear at their goal locations (as it is in Flatland), prioritized planning is complete, i.e. it guarantees finding a solution.

Initially all agents are assigned with the priorities and then the path for each agent is planned in accordance with the imposed priority ordering. Individual planning is performed by the heuristic search algorithm, i.e. A*, that takes time dimension into account. Indeed, when looking for a path for an agent, individual planner has to avoid all the plans already built for high priority agents.

Assigning priorities is very important for prioritized planning as has a vast influence on the algorithm's performance. E.g. 'wrong' priorities may lead to the solution of poor quality and/or getting such solution may take considerably longer time compared to the 'right' priorities). In Flatland we empirically found that faster agents should get higher priorities. It helps them to complete their routes fast and get off the map thus not posing a threat for slower trains.

The most challenging part of Flatland is processing the delays that occur

online (i.e. at simulation). When an agent is delayed due to the malfunction its path became invalid and one needs to alter it.

We start with simply re-planning a path for the delayed agent with considering all other trains as dynamic obstacles with the fixed trajectories (so all trains, except the delayed one, keep sticking to their schedules at first). If the path is not found we remove certain trains from the set of dynamic obstacles one by one and continue re-planning.

To remove the trains we adopt the following ad-hoc strategy that we found (empirically) to work well. We create a queue of the trains that pass through the cell that is occupied by the delayed agent. The queue is sorted by the number of time-steps it takes the train to come to the mentioned cell. Then, a train at the top of the queue is popped out and re-planning for the delayed agent is done. This process repeats until either the plan is found or the queue is empty. In the later case we report *failure* to solve an instance (this does not happen often in practice). In the former case we end up with the delayed train having a valid plan (as well as all the trains that were not put to the queue or were not extracted from there), while some other trains – the one that were popped out of the queue – do not. To fix this, we recursively apply the same procedure to each of the latter trains[1].

On top of that procedure we implement the following improvements that we found to work well and raise the number of handled delays and, consecutively, fraction of done agents. We add extra wait-steps to the agents that appear in the queue more than two times. We re-plan paths for the limited number of agents (i.e. 5) that were close to the local start of the delayed agent but were not affected during the 'fix-the-delay' procedure.

Overall, we were able to achieve 95% success rate (fraction of done agents), which was enough to become the bronze winner of the challenge.

PS: Indeed we are aware of complete and even optimal MAPF solvers, e.g. push-and-rotate, CBS etc., but their computational budget is notably higher compared to the suggested approach.

[1]Please note that such approach does not guarantee solving the problem in general.