Replanning in Advance for Instant Delay Recovery in Multi-Agent Applications: Rerouting Trains in a Railway Hub

Issa K. Hanou[*1], **Devin Wild Thomas**[*2], **Wheeler Ruml**[2], **Mathijs de Weerdt**[1]

[1] Delft University of Technology, The Netherlands
[2] University of New Hampshire, USA
i.k.hanou@tudelft.nl, dwt@cs.unh.edu, ruml@cs.unh.edu, m.m.deweerdt@tudelft.nl

Abstract

Train routing is sensitive to delays that occur in the network. When a train is delayed, it is imperative that a new plan be found quickly, or else other trains may need to be stopped to ensure safety, potentially causing cascading delays. In this paper, we consider this class of multi-agent planning problems, which we call Multi-Agent Execution Delay Replanning. We show that these can be solved by reducing the problem to an any-start-time safe interval path planning problem. When an agent has an any-start-time plan, it can react to a delay by simply looking up the precomputed plan for the delayed start time. We identify crucial real-world problem characteristics like the agent's speed, size, and safety envelope, and extend the any-start-time planning to account for them. Experimental results on real-world train networks show that any-start-time plans are compact and can be computed in reasonable time while enabling agents to instantly recover a safe plan.

Introduction

When executing a multi-agent plan and an agent is delayed, it must reroute to minimize the effect on other agents. In many real-world settings, minor delays happen frequently and can compound into larger holdups that propagate through the network. Thus, replanning must be solved as soon as possible, which is challenging because the search space grows exponentially in the number of agents. Our focus is to swiftly handle holdups that can be resolved by rerouting only the delayed agent. Such a setting is representative of a set of problems that we name Multi-Agent Execution Delay Replanning (MAEDeR) problems. These are single-agent problems that occur during the execution of a multi-agent plan. By quickly reacting without violating the pre-existing plans of other agents, we are more likely to avoid a cascade of delays.

Existing approaches to multi-agent delay recovery usually replan all agents. However, if the delay can be recovered by replanning only the delayed agent, this can be done faster than replanning all agents and requires no communication with the other agents or modifications of their plans. We propose a method for solving MAEDeR by replanning in advance using any-start-time safe interval path planning (@SIPP) (Thomas et al. 2023), which allows for instant delay recovery. The search procedure computes an any-start-time plan for each agent, which is the set of optimal plans for all possible starting times ahead of execution, treating the other agents executing their plans as moving obstacles. An agent can instantly recover a safe plan once its delay is known and execute it without affecting other agents.

Previously, any-start-time planning was used for single-agent problems in grid-based settings. We show how it can be useful in a multi-agent context, specifically railway hub delay replanning. A railway hub is an area with a train station and surrounding shunting yards, which are locations where trains can be parked and serviced. For instance, Fig. 1 shows the Enkhuizen hub with platforms, sidings, and a track connecting it to the railroad network. While we show the example of dense infrastructure hubs, which are the most difficult to plan, the same problem representation is also applicable to larger railway networks. In practice, replanning is still done manually by human operators.

Railway hub delay replanning has several characteristics that are emblematic of real-world applications. First, the agents in these problems are not point agents, because they have a spatial extent, and thus, a temporal extent. Agents occupy several locations simultaneously, such as a 300-meter-long train stretching over two tracks. Because real-world problems rarely have one type of agent, we allow for heterogeneity in size and speed. In a railway hub, switches further constrain the possible moves a train can make. We create a reduction from railway hub delay replanning to an @SIPP graph that inherently encodes the direction of an agent. Finally, we include context-dependent safety measures that agents must respect, like a variable headway (the time between two consecutive trains) that depends on the relative travel directions.

Our main contribution is instantly solving delay response

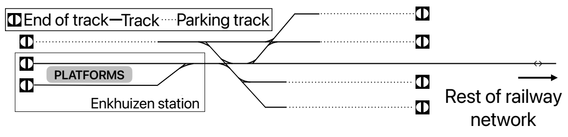


Figure 1: Layout of the Enkhuizen railway hub in the Netherlands [Adapted from SporenPlanOnline].

[*]These authors contributed equally.

in railway hub delay replanning by applying any-start-time planning to multi-agent execution delay replanning. We recover a safe plan to reroute the delayed train without affecting other trains' plans. The precomputation of safe plans allows us to rapidly recover the ability to handle a new delay. We show promising results for handling delays in real-world problems. Our method is also extendable to other multi-agent settings that can be modeled as a MAEDeR problem, for example, routing automated guided vehicles in container terminals.

Example: Railway Hub Delay Replanning

We use railway hub delay replanning to instantiate our methods in an intuitive application. Imagine two trains such as those shown in Fig 2, with two station tracks 1 and 2, a parking track P, two switches 3 and 4, and a track E connecting the station to the rest of the network. Suppose that the initial multi-agent plan is for train I to depart first and traverse its route from I to I' without waiting, clearing the shared track right before train II uses it to travel from II to II'. Because their paths cross over the shared track, if train I is delayed, one of the trains may need to wait in order for both to safely reach their destinations. If train I follows the same plan at a later time, it may conflict with train II. Instead, it should pick a new safe plan that respects train II, such as waiting for train II to clear the shared track, and then finishing its route.

This example follows the Dutch railway operation policies, where trains that are being parked wait for ones serving the timetable. Common causes for delays include too many passengers trying to board, hindrances on the tracks, or a servicing action taking longer than expected. In the following sections, we formalize the intuitions of this example into a planning method to solve this class of problems.

Background

MAEDeR is a single-agent problem in a multi-agent setting. While the classic multi-agent path finding (MAPF) problem constructs non-conflicting plans for a set of agents (Stern 2019), MAEDeR does not come up with the initial multi-agent plan. Instead, it solves the problem of how to quickly recover from one delayed agent while the multi-agent plan is being executed. Delay handling has been previously studied in the context of MAPF, although the focus has been on creating initial plans that are robust to delays (Ma, Kumar, and Koenig 2017; Atzmon et al. 2020a). Where delay recovery has been considered, it has been in the discrete-time context, without singling out a single agent specifically.

Previous studies on delay handling for trains have focused mostly on recovering from disturbances in the complete railway network related to the timetable (Bešinović 2020; Cacchiani et al. 2014). However, railway hub operations differ in that they are more flexible to plan and are planned manually. Some studies have focused on delays solely in a shunting yard. van den Broek, Hoogeveen, and van den Akker (2018) evaluated a set of robustness measures in their scheduling formulation of a shunting plan, using a distribution over activity length to model delays. A different study used a distribution of arrival times to find a plan robust to different

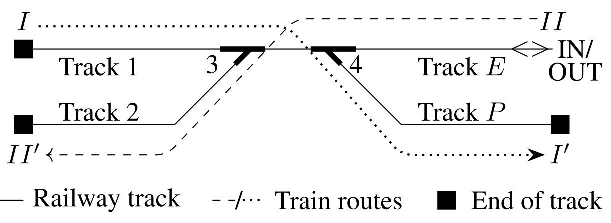


Figure 2: Railway hub delay replanning problem: tracks $(1, 2, E, P)$, switches $(3, 4)$, and trains $(I \to I', II \to II')$.

arrival times, which can be considered delays of the original time (Gardos Reid 2023). Railway hubs have been previously studied as (re)dispatching, including delay response. D'Ariano and Pranzo (2009) used an Alternative Graph formulation with a job shop scheduling approach for conflict detection and resolution to obtain a conflict-free schedule respecting details of the problem formulation (e.g., signals and operational rules). Although their method tries to stick to the original routes, the wait times in the plans of non-delayed trains can be changed. Their method can even cancel or introduce new routes if needed. In contrast, we focus on handling delays without altering the plans of other non-delayed trains. This limits the effects of the delay to only the delayed train and requires no inter-train communication.

Safe Interval Path Planning (SIPP)

Our method for solving MAEDeR relies on the state space of safe interval path planning (SIPP) (Phillips and Likhachev 2011) and the algorithms for solving any-start-time SIPP (@SIPP) (Thomas et al. 2023). A SIPP problem is a single-agent state-space search problem defined by the tuple $\langle S, E, \delta, s_o, x_g \rangle$. A SIPP search state $\langle x, i \rangle \in S$ has two components: x, the configuration (e.g., agent location), and $i = \langle t_s, t_e \rangle$, a *safe interval*, which is a continuous timespan from t_s to t_e when it is safe for the agent to be in configuration x. The edges $\langle u, v, i \rangle \in E$ denote an interval i where the agent can safely transition from configuration u to configuration v. The cost of an edge is its duration $\delta(u, v)$. The objective of a SIPP agent is to find a minimum duration path to the goal configuration x_g starting from the origin state s_o. To solve SIPP problems, Phillips and Likhachev (2011) employ an A* search on this state space, where the objective function $f(s) = g(s) + h(s)$ uses the scalar *earliest arrival time* at SIPP states as $g(s)$. This returns a single safe optimal plan as its solution, arriving as early as possible at each intermediate SIPP state. Using an A* search, the runtime of a SIPP search is $O(|E| \log(|V|))$, where $|V|$ is the number of SIPP search states and $|E|$ is the number of edges in the SIPP graph. Note that several states may share the same configuration, with different non-overlapping safe intervals. Edges between the same pair of states can similarly have several intervals.

Because time is continuous, a SIPP search graph is a compact representation of an infinite problem. For example, take a pedestrian railway crossing. There are two SIPP states: the near side of the crossing and the far side, both of which are always safe. A train passing creates two SIPP edges, the safe

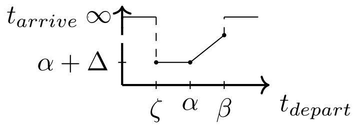


Figure 3: An ATF with parameters ζ, α, β, and Δ.

interval before the train arrives, and the safe interval after it has passed. In this case, a pedestrian could try and cross at any safe time, but the SIPP graph can represent those infinite actions with only two edges.

Any-start-time SIPP (@SIPP)

Recent work on any-start-time SIPP, notated @SIPP (Thomas et al. 2023), describes how to efficiently generate plans for all start times on SIPP graphs. The augmented SIPP (ASIPP) algorithm performs a search that is graphically isomorphic to a SIPP search. However, rather than returning a single plan that arrives at a scalar earliest arrival time, ASIPP returns a family of related paths. All paths in the family move through the same sequence of SIPP states but at different times. ASIPP works by 'augmenting' the SIPP search nodes to track the earliest *arrival time functions* (ATFs) instead of the scalar g values used by SIPP.

The ATF of a path family tells us the earliest arrival time for any departure time along the corresponding sequence of SIPP states. To enable this search, the graph is transformed, from a SIPP graph to an @SIPP graph. For each edge in the SIPP graph, the source, destination, and edge safe intervals are combined into a simple piecewise linear ATF, defined by the parameters $\langle \zeta, \alpha, \beta, \Delta \rangle$ as shown in Fig. 3:

$$f[\zeta, \alpha, \beta, \Delta](t) = \begin{cases} \infty & t < \zeta \\ \alpha + \Delta & \zeta \leq t < min(\alpha, \beta) \\ t + \Delta & \alpha \leq t < \beta \\ \infty & \beta \leq t, \end{cases} \tag{1}$$

where ζ is the earliest time the agent can safely wait at the starting state of the edge, α is the earliest time the agent can safely begin traversing the edge, β is the time the edge becomes unsafe, and Δ is the transit time of the edge.

In a SIPP search, successors are generated with scalar g being the cumulative sum of edge costs along the path so far. In contrast, ASIPP's functional g is the cumulative composition of edge ATFs along the path so far, which maintains the same piecewise-linear structure $\langle \zeta, \alpha, \beta, \Delta \rangle$. However, ASIPP only returns the family of plans that includes the optimal plan at a certain departure time.

Another algorithm developed by Thomas et al. (2023), called RePEAT, repeatedly calls ASIPP and restarts a partial expansion A* search with monotonically increasing start times. RePEAT then compiles the plans returned by ASIPP into a set that can be rapidly queried for an optimal plan corresponding to any departure time. Any-start-time planning can be seen as precomputing a policy for any delay, or similarly as a type of universal plan (Schoppers 1987). Universal planning is "an (almost) universally bad idea" (Ginsberg 1989) because universal plans generally grow exponentially with problem size. However, @SIPP is one of the exceptions: the any-start-time plans grow linearly with problem size (number of safe intervals) (Foschini, Hershberger, and Suri 2014; Thomas et al. 2023), and RePEAT needs to call ASIPP only a linear number of times.

Problem Description: MAEDeR

A Multi-Agent Execution Delay Replanning problem (MAEDeR) is defined by the tuple $\langle N, T, \mathcal{C} \rangle$. The infrastructure network N is a set of connected components that can be represented as a graph with edges between locations. The agents $t \in T$ (i.e., trains) each navigate through the network. The problem characteristics $\mathcal{C}$ define how the agents interact with the network and each other. These are context-specific and include information to calculate the edge duration $\delta(u, v) : \ u, v \in N$ and to constrain when edges can be traversed. We refer to N and T collectively as the system, which is safe if all agents have conflict-free plans.

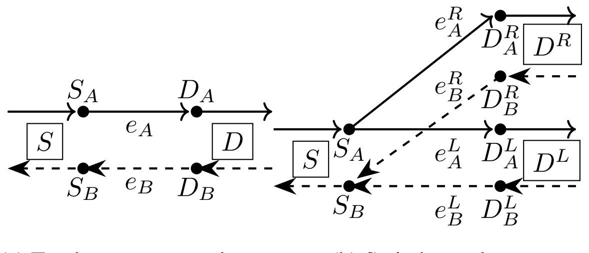
A solution to MAEDeR is a function $\mathcal{F}$ taking an agent $a \in T$ and a positively delayed start time d and returning the shortest safe plan for the delayed agent, or a failure if no feasible plan exists. The returned plan does not require modifications of the plans of any agent other than the one that was delayed. When an agent is delayed, the system is no longer safe until the function returns a new plan for this agent. We refer to this period as the *interval of uncertainty*. The objective of MAEDeR is to provide a solution that minimizes the interval of uncertainty.

Specifics for Railway Hub Delay Replanning

We now illustrate the MAEDeR definition for railway hub delay replanning. In a railway hub, the components of the network N include track segments, platforms, and switches. The trains moving through the hub are the set of agents T. The characteristics $\mathcal{C}$ include each train's length λ and speed ν, and the length of each track segment $\ell(u, v)$. The duration to traverse an edge can be calculated as $\delta(u, v) = \frac{\ell(u,v)}{\nu(a)}$.

The constraints for traversing edges in a railway hub relate to the static hub infrastructure and movements of trains. As trains cannot navigate sharp angles, they must be reversed to change direction. To do so, the driver has to walk to the other side of the train. Therefore, we need the driver's walking speed $\omega \in \mathcal{C}$. The constraints also determine when edges can be traversed to avoid conflicts and construct a safety envelope for each train, which is a safety buffer between trains. An action's safety envelope is defined by a train's *headway*, which is the time between two consecutive trains. This is context-specific: we define the *following headway* $\epsilon_f \in \mathcal{C}$ for trains traveling in the same direction and the *crossing headway* $\epsilon_c \in \mathcal{C}$ for the opposite direction. Other MAEDeR problems may share some or all characteristics with railway hub delay replanning.

One such problem is railway freight traffic planning, which is done ad hoc, depending on the arrival of supply. Since the start and destination are known in advance, we can precompute the any-start-time plan of a freight train. A route can be queried once a train is ready to depart. So, our method can be used out of the box for these scenarios, too.



(a) Track segment topology. (b) Switch topology.

Figure 4: Topology of (a) track segments and (b) switches.

When MAEDeR Fails

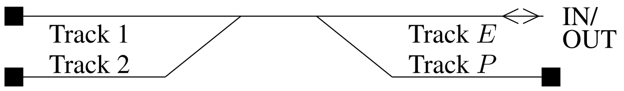
The MAEDeR solution function $\mathcal{F}$ calculates a feasible shortest plan for agent a starting at delayed time d. If no safe plan with start time d exists, then $\mathcal{F}$ returns that no such plan exists. How the system reacts to this is problem-specific and outside the scope of this paper. An example reaction could be to execute a multi-agent replanning algorithm. Measures to ensure safety in the meantime are also problem-specific. For example, when planning driverless taxis, this could be an all-stop order halting all taxis until a new safe multi-agent plan is found.

Reducing MAEDeR to Any-Start-Time SIPP

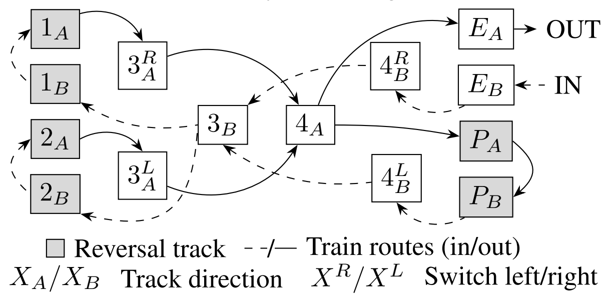
To solve MAEDeR problems, we provide a reduction to @SIPP. In short, for each agent, we transform the network into a SIPP graph with safe intervals on the states and edges for that agent, treating the other agents as moving obstacles. We then compute an any-start-time plan for that agent, so that when it is delayed, it can instantly recover a safe plan. From here on, we demonstrate this reduction for the railway hub delay replanning problem, but this can also be applied to other MAEDeR contexts, such as navigating automated vehicles in container terminals or dense road networks.

First, we give an intuition on this reduction. The locations in the transformed graph loosely correspond to points in the railway network where track segments meet or join other infrastructure components. The actual track segments are represented by edges, one per possible travel direction. The unsafe intervals result from other trains moving through the network, blocking safe access to the track(s) that they occupy. This reduction must obey the physical properties of the infrastructure, and maintain the safety of the network.

In railway hub delay replanning problems, these properties (characteristics $\mathcal{C}$) are inherent to tracks and switches. Because trains move forward on the track and cannot simply turn around, the available edges for a train to follow depend on the direction it is traveling. For example, in Fig. 4a, two straight track segments next to each other are connected: S on the left and D on the right. A train can go either left to right or right to left, based on its initial direction. So, we split each location into two co-located states (A and B) as is the custom in the railway sector. Now, when going from the left track to the right track, the train goes from S_A to D_A. The other way around, a train would go from D_B to S_B. We transform the undirected edges of the train network into



(a) Layout from Fig. 2.



(b) Graph representation of the layout.

Figure 5: Modeling the layout graph (solid versus dashed lines are used to demonstrate the direction of a route).

pairs of antiparallel directed edges, where A-sides of states have directed edges solely to A-sides, and B-sides only to B-sides. The A and B-sides are only connected if trains can reverse on this track, like at the end of track 1 in Fig. 5a.

To model a switch, we use three pairs of co-located states, as shown in Fig. 4b. Here, we see the conjunction of three tracks, with S on the left and two tracks D on the right. The two tracks are named D^L and D^R to signify the left and right sides of a switch, also a custom in railway operations. A train incoming from track S encounters location S^A, which has two successors D_A^L and D_A^R. Oppositely, a train coming from either track D^L or D^R will use the B-side and continue its route over point S^B. Since a switch always forms an acute angle between two tracks on the same side, a train cannot make that turn, so the same-side nodes are not connected.

These translations are used to construct the graph of a hub railway layout. An example is shown in Fig. 5. All four tracks $(1, 2, E, P)$ have two nodes (A/B-sides) and switches have additional R/L-nodes on the same side of the switch. We see that only the reversal tracks $(1, 2, P)$ have their A and B-sides connected. Finally, all A-sides are connected to neighboring A-sides, and similarly for the B-sides.

Safe Interval Generation

Given the graph of the network infrastructure N, we want to generate safe intervals that allow our agent to navigate safely given the other agents in the problem. We now show the interval generation process specifically for railway hubs, though this can be similarly done for any type of obstacles moving across an infrastructure. We start by tracing out the unsafe intervals created by other trains, which are then inverted to form safe intervals. Intuitively, a location is unsafe when it is occupied by (part of) a train, and an edge is unsafe when a train occupies the start of it. We have co-located states for each location, and safe intervals are often shared between them. The antiparallel edges joining a track's two pairs of co-located states have a more complicated relationship. An edge that is traversed by a train is unsafe until the train has completely departed the edge's origin. On the other hand, the antiparallel edge is unsafe while the train is still occupying any part of the edge.

We calculate the unsafe intervals as follows. Take agent train a traveling from location u_A to v_A beginning at t_0. The time to traverse the edge $\delta(u,v)$, and the duration δ_a of a passing one point on the edge are

$$\delta(u,v) = \frac{\ell(u,v)}{\nu(a)}, \delta_a = \frac{\lambda(a)}{\nu(a)}. \tag{2}$$

The time to fully traverse the edge front to rear is thus $\delta(u,v) + \delta_a$. The unsafe interval for location u_A is

$$i_{u_A} = \langle t_s^{u_A}, t_e^{u_A} \rangle = \langle t_0, t_0 + \delta_a + \epsilon_f \rangle, \tag{3}$$

where train a arrives at u_A at t_0, the duration to traverse u_A is δ_a, and we add the headway ϵ_f to complete the safety envelope. For location v_A, the unsafe interval is

$$i_{v_A} = \langle t_s^{v_A}, t_e^{v_A} \rangle = \langle t_0 + \delta(u,v), t_s^{v_A} + \delta_a + \epsilon_f \rangle, \tag{4}$$

where the start time of the interval is the moment the train arrives at location v_A (start time t_0 + traversal $\delta(u,v)$), and the end of the interval has the additional duration δ_a and headway ϵ_f. For the end of the intervals, we need the time for the rear of the train to depart, which is the length of the train over its speed. Co-locations u_B, v_B are unsafe in the intervals

$$i_{u_B} = \langle t_s^{u_B}, t_e^{u_B} \rangle = \langle t_s^{u_A}, t_s^{u_A} + \delta_a + \epsilon_c \rangle, \tag{5}$$

$$i_{v_B} = \langle t_s^{v_B}, t_e^{v_B} \rangle = \langle t_s^{u_B} + \delta(u,v), t_s^{v_B} + \delta_a + \epsilon_c \rangle, \tag{6}$$

so the only difference is the headway ϵ_c that is included instead of ϵ_f. The edge $e = (u_A, v_A)$ is unsafe from

$$i_e = \langle t_s^e, t_e^e \rangle = \langle t_s^{u_A}, t_e^{u_A} \rangle, \tag{7}$$

which already includes the headway as well. The antiparallel edge $e' = (v_B, u_B)$ has the unsafe interval

$$i_{e'} = \langle t_s^{e'}, t_e^{e'} \rangle = \langle t_s^{u_A}, t_e^{v_A} \rangle. \tag{8}$$

For example, consider again the scenario in Fig. 2, where a train II is routed from E_B to 2_A. We construct the unsafe intervals for train II, which are shown in Fig. 6, using the parameters given in Table 1. Take train II traveling from location E_B to location 4_B^R which takes $\delta(u,v) = 100$ (Eq.2). The train departs E_B at $t_0 = 100$ and its front arrives at location 4_B^R at 200 (Eq. 4). The end of the unsafe interval for location E_B is 260, which adds the time for the rear to leave (60) and the headway (100) to the start $t_0 = 100$ (Eq. 3). The end of the unsafe interval for location 4_B^R is 360 (Eq. 4). This yields the unsafe intervals $\langle 100, 260 \rangle$ and $\langle 200, 360 \rangle$ for locations E_B and 4_B^R, respectively. The co-locations $\langle E_A, \langle 100, 210 \rangle \rangle$ and $\langle 4_A, \langle 200, 310 \rangle \rangle$ have similar intervals with only the headway difference $\epsilon_f - \epsilon_c$ (Eq. 5 and 6).

For reversal tracks, the intervals are calculated a bit differently. As mentioned, we must ensure that the train has enough time to be reversed. So, when a train arrives at a dead-end track, there is a buffer time before the internal edge becomes a safe action. This time is calculated as the time for the driver to walk across, using driver speed ω and train length $\lambda(a)$. The unsafe interval for such a location y is

$$i_y = \langle t_s^y, t_e^y \rangle = \left\langle t_s^y, t_s^y + \frac{\lambda(a)}{\omega} + \epsilon_f \right\rangle, \tag{9}$$

where both co-locations y_A, y_B of the track get the same interval, as well as the edge (y_A, y_B). For example, the locations 2_A and 2_B have such an interval in Fig. 6.

For a switch, consider the topology shown in Fig. 4b. Take a train traveling from S_A through the switch to D_A^R. The switch should be unsafe to traverse while the train is moving through it, but it should be safe for a train to wait at D_B^L for the switch to clear. The unsafe intervals for S_A and S_B can be calculated using Equations 3 and 5, respectively, and the same for D_A^R (Eq. 4) and D_B^R (Eq. 6). Since state D_A^L is co-located with D_A^R they share the same interval. State D_B^L has no unsafe interval for this move, because a train could technically wait here. For the edges, edge (S_A, D_A^R) has the interval $\langle t_s^{S_A}, t_e^{S_A} \rangle$ (Eq. 7) and edge (D_B^R, S_B) has interval $\langle t_s^{S_A}, t_e^{D_A^R} \rangle$ (Eq. 8). Since there is in practice only one track part that is the switch, the edges all share the same intervals. So, the intervals for (S_A, D_A^R) and (S_A, D_A^L) are the same based on Equation 7 and the intervals for (S_A, D_A^L) and (S_A, D_A^L) are equal based on Equation 8. The example (Fig. 6) shows the resulting intervals for switches 3 and 4.

Following this reduction, we have a SIPP problem with safe intervals on states and edges. We can apply the approach of Thomas et al. (2023) to compile the safe intervals into edge arrival time functions, and then solve it as an any-start-time SIPP problem.

t_0	$\lambda(t)$	$\nu(t)$	ω	ϵ_f	ϵ_c
100s	600m	10 m/s	1 m/s	100s	50s

Table 1: Information for scenario in Fig. 6, train II (t).

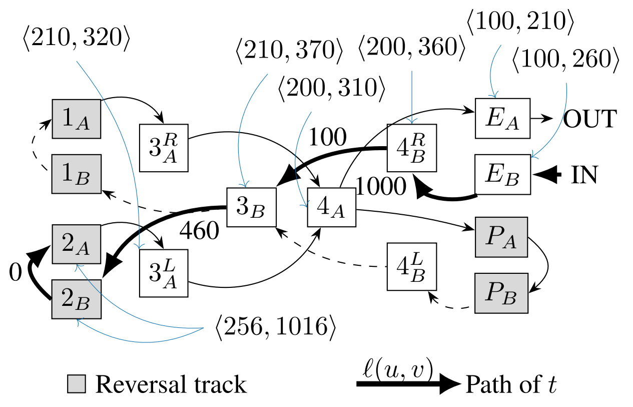


Figure 6: Unsafe intervals for train II (t) from Fig. 2. These form the obstacles for train I to navigate through. For each node involved in the move, the associated intervals are given.

Solving MAEDeR

The generic planning loop for solving MAEDeR consists of the following points in time we call milestones:

Unsafe when a delayed agent learns it is delayed,

Solve the MAEDeR function is applied,

Safe when the delayed agent regains a safe plan,

Recompute a new MAEDeR function is derived,

Recovered when the system can handle a new delay.

The interval of uncertainty is between unsafe to safe. An effective method solving MAEDeR minimizes the uncertainty interval. Moreover, it minimizes the time to recompute a new solution, allowing sooner handling of a second delay.

We describe two algorithms for solving MAEDeR: replanning SIPP (rSIPP) runs a new SIPP search for the delayed agent, while any-start-time planning for MAEDeR (@MAEDeR) queries an any-start-time plan. The rSIPP solution consists of a set of SIPP graphs. Each agent has a corresponding SIPP graph with safe intervals where all other agents are treated as moving obstacles. When a delay is encountered, rSIPP selects the SIPP graph of the delayed agent and runs a SIPP search to find a new safe plan starting after the delay. The solving milestone for rSIPP searches the precomputed SIPP graph, while the recomputation milestone precomputes the SIPP graphs.

@MAEDeR trades increased precomputation time for eliminating the interval of uncertainty. The precomputation of @MAEDeR generates the same set of SIPP graphs for each agent as rSIPP. These graphs are then transformed into @SIPP graphs, and we replan in advance using RePEAT to compute the any-start-time plan for each agent as part of the precomputation. The @MAEDeR solution is the set of every agent's any-start-time plan. When an agent is delayed, the solving milestone for @MAEDeR queries their any-start-time plan, while the recomputation milestone precomputes the @SIPP graphs given that new plan and runs the RePEAT searches.

Querying an any-start-time plan is logarithmic in its size, which is at most linear in the number of safe intervals in the SIPP graph. Each location in the graph adds a safe interval to the problem, and each agent splits the safe intervals it passes through, adding one interval to each state and/or edge it traverses. As such, the problem size scales linearly with the number of locations and number of agents. The first step of @MAEDeR's recomputation is to generate the SIPP graphs needed for rSIPP, meaning that @MAEDeR can fall back to rSIPP if another delay happens before it has finished recomputing the any-start-time plans. In fact, MAEDeR will reach rSIPP's recovery point before rSIPP would have, because its interval of uncertainty is shorter.

$\lambda(t)$	$\nu(t)$	ω	ϵ_f/ϵ_c
100-2000	5-50	0.5-5	50-500

Table 2: Sampled values for scenario generation $\forall t \in T$.

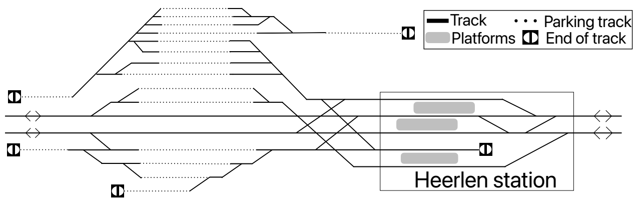


Figure 7: Layout of the Heerlen railway hub in the Netherlands [Adapted from SporenPlanOnline].

Experimental Evaluation

The goal of our experiments is to empirically evaluate the performance of rSIPP and @MAEDeR, including our claim that @MAEDeR is 'effectively instant'. We want to answer the following question:

Q1 Is the interval of uncertainty for @MAEDeR significantly shorter than for rSIPP?

We also demonstrate the appeal of both methods in practice:

Q2 Are the recomptutation times for both methods using realistic scenarios reasonable for practical application?

Data

We created our own dataset based on real-world data from two Dutch railway hubs. The smaller one is based on the station of *Enkhuizen* (Fig. 1) which has a total of eight tracks. The larger one with 29 tracks is the station of *Heerlen* shown in Fig. 7, which also has 'free tracks' that can be entered from both sides. The hub layouts comprising the network N were computed manually based on the information available online[1], taking the length of tracks in meters. We assign all platforms and parking tracks to be places where trains can turn around. Intermediate track segments or ongoing tracks (like IN/OUT in Fig. 5) do not allow trains to reverse. This way, trains are not allowed to stop in the middle of a track where other trains can still be traveling.

For the Enkhuizen hub, we created three scenarios. The small scenario (6 trains) was constructed manually and the medium scenario (13 trains) is based on the actual timetables showing the necessary moves on a Tuesday morning (October 31, 2023)[2]. This scenario uses realistic headway times (Liu and Han 2017; Wang, Liu, and Zeng 2017), train speeds, and train lengths. Finally, we generated a large scenario of 25 trains (more is unrealistic as the Enkhuizen hub does not have enough tracks for that many trains). For the Heerlen hub, we also generated scenarios with 6, 13, and 25 trains for comparison. Additionally, as this layout is much bigger, we created a scenario with 50 trains. For each scenario, we created different instances by assigning a different train as our agent, so we had variations of the same scenario.

The scenario generation samples several values using a given random seed (see Table 2). Each train gets a set of routes, which define start and end locations, and there can be either 1, 2, or 3 ordered subgoals for the train to reach. The

[1]sporenplan.nl, openrailwaymap.org

[2]ns.nl/reisplanner, treinposities.nl, treinenweb.nl/materieel

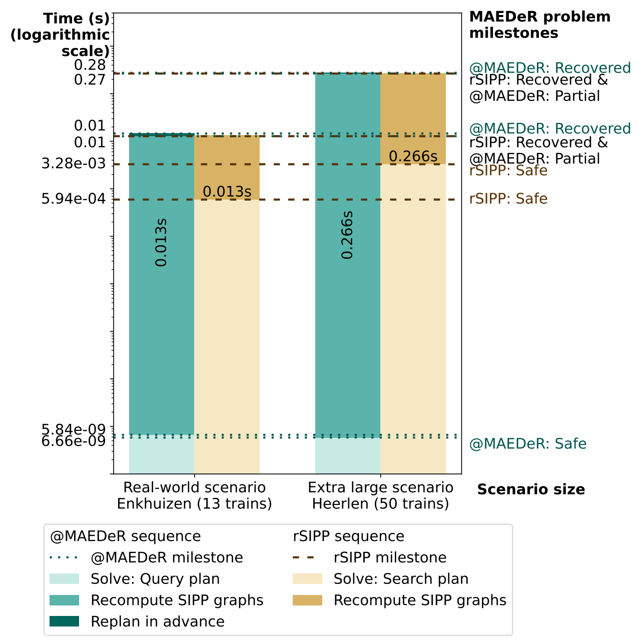


Figure 8: Comparing the average time to reach the milestones described in Section Solving MAEDeR.

start time of a route is sampled from an interval of $\langle 0, 1000 \rangle$. We find the shortest path through the time-dependent state space for each route (i.e., time-independent route between start and goal location), and the end time of the route is calculated using the agent's sampled speed. Successive trains are generated with progressively increasing start times. If the trains in the scenario have no conflicts, then this route is included in the scenario and we move on to the next train route. Otherwise, a new route is sampled until we have the number of required routes for the total number of trains.

Implementation

The code to replicate our experiments is available on GitHub[3]. We ran our experiments using an Apple M1 Pro processor. The **Q1** and **Q2** search results use an implementation of rSIPP and RePEAT in an efficient C++ code base, using the same search code and data structures when possible. The **Q2** graph computation results currently use an implementation of SIPP and @SIPP graph generation in Python. The search timeout was set to 300s, but never reached.

Results and Discussion

Fig. 8 compares the two algorithms in their time to reach the milestones described in the Solving MAEDeR section. The 'Recompute SIPP graphs' step is the same procedure for both algorithms and thus takes an equal amount of time, although it appears different due to the log scale (the time is also shown in the respective boxes). The 'Replan in advance' step ('@MAEDeR recovered' in Fig. 8) recomputes the any-start-time plans for all other agents, so every agent

[3]https://github.com/dwthomas/delay-replanning

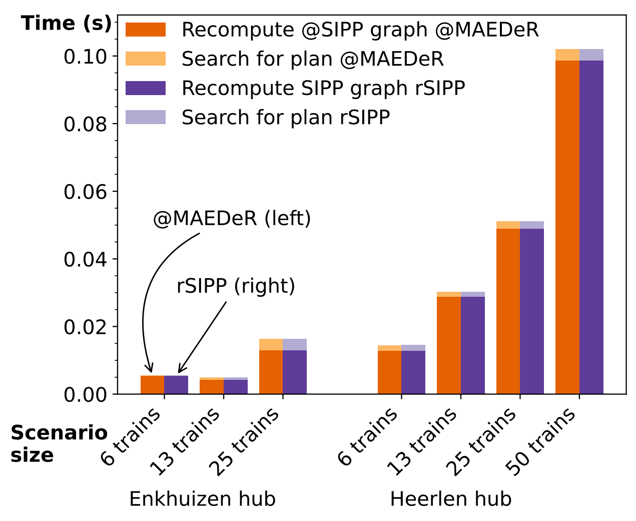


Figure 9: Average replanning runtime of the algorithm for different size scenarios on the Enkhuizen layout.

is ready to query their updated any-start-time plan in case of a new delay. To answer **Q1**, we measured the interval of uncertainty, which is the time to reach the **Safe** milestone. In Fig. 8, we can clearly see a significant difference between the algorithms, @MAEDeR's interval of uncertainty is tens of nanoseconds, while rSIPP takes almost 0.1 milliseconds for the largest scenario, which is 1000 times slower. We find that @MAEDeR reaches the safe milestone $\sim$1 ms before rSIPP does.

This result provides empirical support for our assertion that looking up an any-start-time plan is effectively instant. Theorem 1 shows that the interval of uncertainty is over before the other trains ever knew it began.

Theorem 1. *In MAEDeR problems of the scale of railway hub delay replanning, @MAEDeR ends the interval of uncertainty before any other agent can be informed of the delay.*

Proof. Our empirical results show that @MAEDeR ends the interval of uncertainty within tens of nanoseconds. Trains within train hubs are generally separated by more than tens of meters. The speed of light is $\approx$0.3 m/ns, and information cannot travel faster than the speed of light. Thus, the delayed agent has ended the interval of uncertainty before any other agent could physically receive word of its delay. $\square$

For **Q2**, we compare the recomputation runtimes of both methods for different scenario sizes in Fig. 9. We see that the recomputation time is only tenths of a second, in which it is very unlikely that a new delay is already problematic, taking into account that the headway between trains is often 2 minutes in practice. As the scenario sizes are representative of real-world scenarios for these hubs, both methods offer attractive runtimes. So, agents can quickly recompute their plans to allow a new delay to be handled. These attractive runtimes would only improve with a more performant graph generation implementation.

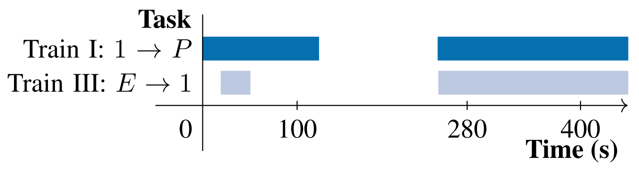


Figure 10: Safe intervals for train I from Fig. 2 and train III.

From Fig. 9, we can see that the runtime scales linearly in the problem size, which is quadratic in the number of trains because the runtime is the sum of all trains in the scenario. The runtime for a single agent is linear in the total number of trains and the number of locations, which explains the difference between the two hubs.

Finally, we consider the additional benefits of the @MAEDeR solution. Our results showed that about 80% of all delays are resolved by maintaining the same route while departing at a later time. In other cases, we reroute the delayed train along a different set of tracks. Besides replanning delayed trains, our method can also visualize the safe intervals, which can help human operators when planning ad hoc freight traffic. Train paths are the regular routes that trains can make, so for any new train that has to be planned ad hoc, we can quickly look up when this train can safely make its route. In Fig. 10, Train I's intervals show when it can move around train II, and the new train III's intervals give its possible plans around train I and II from Fig. 2. Train III represents a common scenario in real-world railway hub delay replanning, especially when considering freight traffic. These trains are often dependent on the arrival time of a container ship, so they need to be planned last minute.

Related Work

MAPF with Delay Response considers the delays of agents by predicting these upfront and handling them during plan execution (Ma, Kumar, and Koenig 2017). The effectiveness of this approach obviously depends on the accuracy of the predictions. A general framework for MAPF that is more robust against delays is called k-robust MAPF (Atzmon et al. 2020a) and extends the safety envelope for a grid node with k time steps. However, a delay suffered by one agent is often propagated to other agents, something we wish to avoid. One solution to this uses a temporal network for post-processing, but delays are still handled ad hoc (Hönig et al. 2016). This can be a complex task to solve and we wish to avoid lengthy computations while the system is in an unsafe state, so a precomputed recovery plan is safer.

There has been much work on speeding up trajectory planning in dynamic environments (Nantabut 2023, inter alia). However, we wish to avoid planning from scratch when a delay is encountered. MAPF with heterogeneous agents has been proposed before (Atzmon et al. 2020b) and a MAPF model was developed specifically for train routing (Švancara and Barták 2022), which also includes the length of a train agent, but does not allow for different agent speeds. Similarly, the Flatland challenge also addressed train routing from a MAPF perspective and included delays (Mohanty et al. 2020). While they assume a grid world structure, railway hubs have a more constrained infrastructure, adding complexity to the problem. However, their focus is on allocating tracks for the timetable, not a railway hub, and they do not include the dynamics of the environment. All the MAPF works cited here handle delays by either predicting them, planning such that they are less likely to occur, or planning ad-hoc in response, while the main benefit of our method is precomputing new plans for possible delays. As we do not replan from scratch, our method relates to plan repair in multi-agent plan execution (Komenda, Novák, and Pěchouček 2012). As the agent can safely execute its precomputed plan, it only needs to send an update of its new plan to other agents, so they can precompute their set of plans, leading to minimal communication.

Conclusion

This paper provides a solution for handling delayed trains in a railway hub. When an agent is delayed, it can instantly recover a safe plan and resume execution, eliminating the interval of uncertainty during which the railway hub system is unsafe. More generally, we demonstrated how to apply any-start-time planning to a multi-agent real-world setting. We defined the multi-agent execution delay replanning problem (MAEDeR) and showed how to transform it into safe interval path planning (SIPP). This can then be used with an any-start-time planning approach to precompute safe plans for agents. Because these plans can be computed prior to execution, the agent can immediately use its safe plan once its delay is known.

Compared to earlier work on any-start-time SIPP, our approach extends beyond the grid-based pathfinding domain. Moreover, we allow for different agent sizes and speeds, we let the agent spatially occupy several locations, and we inherently encode the movement direction in our graph. The latter is especially important in MAEDeR problems where the agents cannot easily turn around.

The experiments showed that the lookup time for a safe plan is instantaneous, so agents can recover their safe plan and immediately execute it without extra waiting time. Furthermore, we show that the use of a SIPP graph with safe intervals still allows for fast replanning, enabling other agents to quickly react to one delayed train. Our method scales well, solving real-world size scenarios in a reasonable time, allowing operators to respond rapidly. Moreover, the approach can also be used for the ad hoc planning of new trains, which is very common for freight traffic. Our approach can be applied to other MAEDeR problems, like moving automated guided vehicles in a container terminal or navigating self-driving cars in dense urban areas.

Acknowledgements

This work is part of the NWO LTP-ROBUST RAIL Lab, a collaboration between the Delft University of Technology, Utrecht University, NS, and ProRail. More information at https://icai.ai/icai-labs/rail/. We are also grateful for support from the NSF-BSF program via NSF grant 2008594.

References

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2020a. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, 67: 549–579.

Atzmon, D.; Zax, Y.; Kivity, E.; Avitan, L.; Morag, J.; and Felner, A. 2020b. Generalizing Multi-Agent Path Finding for Heterogeneous Agents. In *Thirteenth Annual Symposium on Combinatorial Search*.

Bešinović, N. 2020. Resilience in railway transport systems: a literature review and research agenda. *Transport Reviews*, 40(4): 457–478.

Cacchiani, V.; Huisman, D.; Kidd, M.; Kroon, L.; Toth, P.; Veelenturf, L.; and Wagenaar, J. 2014. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63: 15–37.

D'Ariano, A.; and Pranzo, M. 2009. An Advanced Real-Time Train Dispatching System for Minimizing the Propagation of Delays in a Dispatching Area Under Severe Disturbances. *Networks and Spatial Economics*, 9(1): 63–84.

Foschini, L.; Hershberger, J.; and Suri, S. 2014. On the Complexity of Time-Dependent Shortest Paths. *Algorithmica*, 68(4): 1075–1097.

Gardos Reid, R. 2023. *Inferring Robust Plans with a Rail Network Simulator*. MSc thesis, Delft University of Technology.

Ginsberg, M. L. 1989. Universal planning: An (almost) universally bad idea. *AI magazine*, 10(4): 40–40.

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*, 477–485.

Komenda, A.; Novák, P.; and Pěchouček, M. 2012. Decentralized Multi-agent Plan Repair in Dynamic Environments. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. AAMAS.

Liu, P.; and Han, B. 2017. Optimizing the train timetable with consideration of different kinds of headway time. *Journal of Algorithms & Computational Technology*, 11(2): 148–162.

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*.

Mohanty, S.; Nygren, E.; Laurent, F.; Schneider, M.; Scheller, C.; Bhattacharya, N.; Watson, J.; Egli, A.; Eichenberger, C.; Baumberger, C.; Vienken, G.; Sturm, I.; Sartoretti, G.; and Spigler, G. 2020. Flatland-RL : Multi-Agent Reinforcement Learning on Trains. arXiv:2012.05893.

Nantabut, C. 2023. *A*-based trajectory planning in dynamic environments for autonomous vehicles*. Ph.D. thesis, Rheinisch-Westfälische Technische Hochschule Aachen University.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation*.

Schoppers, M. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 87, 1039–1046.

Stern, R. 2019. Multi-agent path finding–an overview. *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*, 96–115.

Thomas, D. W.; Shimony, S. E.; Ruml, W.; Karpas, E.; Shperberg, S. S.; and Coles, A. 2023. Any-Start-Time Planning for SIPP. In *ICAPS Workshop on Heuristics and Search for Domain-Independent Planning*.

van den Broek, R.; Hoogeveen, H.; and van den Akker, M. 2018. How to Measure the Robustness of Shunting Plans. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65, pp 3:1–3:13. Dagstuhl, Germany: Open Access Series in Informatics (OASIcs).

Švancara, J.; and Barták, R. 2022. Tackling Train Routing via Multi-agent Pathfinding and Constraint-based Scheduling. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*, 306–313. SciTePress.

Wang, G.; Liu, H.; and Zeng, X. 2017. Study on train headway in different turning-back mode of urban mass transit station. *Transportation Research Procedia*, 25: 451–460.