Precomputing Multi-Agent Path Replanning Using Temporal Flexibility

Issa Hanou[1], **Eric Kemmeren**[1], **Devin Wild Thomas**[2], **Mathijs de Weerdt**[1]

[1]Delft University of Technology
[2]University of New Hampshire
{i.k.hanou, m.m.deweerdt}@tudelft.nl

Abstract

Executing a multi-agent plan can be challenging when an agent is delayed, because this typically creates conflicts with other agents. So, we need to quickly find a new safe plan. Replanning only the delayed agent often does not yield an efficient plan, and sometimes cannot even yield a feasible one. On the other hand, replanning other agents may lead to a cascade of changes and delays, and it is computationally expensive. We show how to efficiently replan a single delayed agent by tracking and using the *temporal flexibility* of other agents while avoiding cascading delays. This flexibility is the maximum delay that the agent can take without changing the order with agents other than the initially delayed agent, or further delaying other agents. Our algorithm, FlexSIPP, precomputes all possible plans for the delayed agent and returns the changes to the other agents within the given scenario. We demonstrate our method in a real-world case study of replanning trains in the densely-used Dutch railway network and in the MovingAI MAPF benchmark set. Our experiments show that FlexSIPP provides effective solutions relevant to real-world adjustments, and within a reasonable timeframe.

Code — https://doi.org/10.5281/zenodo.20543671

Introduction

When executing a plan in a multi-agent environment, delays frequently occur, leading to conflicts and disrupting the plan's previously safe execution. Resolving these conflicts requires deciding which agents' plans to disrupt and by how much. As conflicts compromise the safe execution, a decision must be made swiftly, while replanning takes time.

In this paper, we address the multi-agent path replanning problem, where a single agent needs to be replanned (e.g., due to a delay), and our objective is to recover a safe plan as quickly as possible. Full replanning of all agents can be computationally expensive, so we consider the subset of these problems where other agents may only be delayed, not rerouted or re-ordered, and this delay must not introduce additional conflicts. These constraints are inspired by the railway network application. We quantify an agent's ability to absorb a delay as its *temporal flexibility*, which is the maximum delay it can take without changing the order in which other agents use a resource or further delaying them. The temporal flexibility is easily precomputed from a given plan. This allows us to identify the set of *tipping points* for the delayed agent, which are the time points until which the delayed agent can use another agent's temporal flexibility and thus go first, and after this tipping point, the other agent should go first, and the ordering of the agents thus changes. FlexSIPP also provides a second use case: given a single delayed start time, return a plan minimizing the total delay.

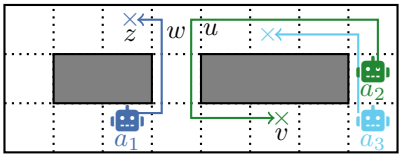

Figure 1: Example Multi-Agent Pathfinding warehouse problem. Each corridor has a width of one.

Figure 1 shows a delay-replanning problem example with three agents starting at $t = 0$ that must arrive before $t = 12$ at their goal. While this shows a discretized space and time, the model and algorithms are continuous. Agents a_1 and a_2 both use the center corridor, where agent a_1 is set to go first. Suppose that agent a_1 is delayed for departure at $t = 0$, but we do not know how much. Agent a_2 is set to enter the corridor at $t = 5$ and exit at $t = 8$, so agent a_1 can no longer use the corridor before agent a_2 without delaying it. Agent a_2 has three time steps of flexibility (arrive by $t = 12$) from location u on, because before that a_3 is right behind and should not be affected. Departing by $t = 1$, agent a_1 can cross the corridor as planned. When departing during $t \in [1, 4\rangle$, agent a_1 can go first, so the order remains the same because agent a_2 can use its flexibility at u departing there at $t = 8$ (makespan 12). After $t = 4$, agent a_1 can wait for agent a_2 and then use the middle corridor, departing at $t = 7$ (makespan 12); or agent a_1 can reroute through the left corridor, which is a longer path, so this is only faster when departing during $t \in [4, 5\rangle$. This example shows the optimal routes based on the departure time of agent a_1, resulting in the tipping point at $t = 4$: if agent a_1 leaves before this time, then it can go first (possibly using a_2's flexibility); otherwise agent a_2 should go first (and agent a_1 may reroute through the left corridor).

This work is closely related to Multi-Agent Execution Delay Replanning (MAEDeR) (Hanou et al. 2024), which addresses a stricter subset of delay replanning problems, where the originally delayed agent must recover without affecting the plans of any other agents. Like MAEDeR, our approach uses the Safe Interval Path Planning (SIPP) state space to encode when resources are not used by other agents (Phillips and Likhachev 2011). With this problem representation, optimal single-agent plans for all start times can be efficiently computed (Thomas et al. 2023), by treating other agents as moving obstacles. Hanou et al. (2024) used this to shift replanning to be 'in advance', and recover from some execution conflicts instantly by looking up a new plan for the delayed agent in its any-start-time plan. While the resulting multi-agent plan is safe, MAEDeR does not necessarily provide an optimal solution, as it assumes that the plans of agents other than the delayed agent cannot be modified.

Compared to MAEDeR and SIPP-based replanning, because we allow delaying other agents, we can reduce overall delay. Our method, called FlexSIPP, still provides near-instant delay replanning, and the required precomputation runs in polynomial time. FlexSIPP was originally described by Kemmeren (2025) as a railway-specific approach. In this paper, we formalize FlexSIPP and generalize it to a multi-agent, continuous-time, path-replanning setting, showing that we can efficiently replan a single agent when it is delayed or its initial plan fails for other reasons. FlexSIPP provides two use cases: precompute the any-start-time plan for any delayed agent to determine the tipping point, or quickly handle a single delay by returning a plan that minimizes total delay. If only delaying agents with flexibility does not yield a plan, we quickly determine so and require full replanning.

Our main contribution is the FlexSIPP algorithm, which quickly replans a delayed agent in a multi-agent pathfinding context by sequentially handling delays. We present a method to determine agents' flexibility and use tipping points to illustrate the replanning trade-off between delaying other agents and further delaying or rerouting a delayed agent. We evaluate FlexSIPP on the MovingAI MAPF benchmark and a case study of the Dutch railway network, demonstrating the benefits of automatically identifying tipping points and using them in real-world situations.

Background

We begin by describing the state-space representation of Safe Interval Path Planning (SIPP); Any-Start-Time Planning, which is the single-agent optimal search used by our precomputation; and Multi-Agent Execution Delay Replanning (MAEDeR), which is the most closely related formulation of the multi-agent delay replanning problem.

Safe Interval Path Planning

SIPP (Phillips and Likhachev 2011) is a single-agent state-space search problem defined by a tuple $\langle S, E, \delta, x_o, t_o, x_g \rangle$, where each state $s = \langle x, i \rangle \in S$ is a configuration x (e.g., agent location) and a *safe interval* $i = [t_s, t_e\rangle$, which is a continuous timespan from t_s to t_e when it is safe for the agent to be in configuration x. An edge $e = \langle s_1, s_2, i \rangle \in E$ for states $s_1, s_2 \in S$ denotes the safe interval i when the edge (s_1, s_2) can be used by the agent, and the cost of this edge is defined by $\delta(s_1, s_2)$, which is the minimum time it takes an agent to traverse the edge. A SIPP graph is thus similar to a time-expanded graph, representing all intervals during which edges can be safely traversed. The objective of a SIPP problem is to find the path of minimal duration δ from the starting configuration x_o at the starting time t_o to a state with the goal configuration x_g. One configuration may have many SIPP states with different (non-overlapping) safe intervals. A SIPP state can also have several SIPP edges connected to another SIPP state. For example, in the scenario in Figure 1, agent a_1 has one SIPP state in its final configuration z with an infinite interval, but the previous configuration w on its path originally has two SIPP states because this configuration is only safe for a_1 to be in before or after agent a_2 crosses it. Both these states connect to the SIPP state on configuration z with a different SIPP edge, allowing agent a_1 to move from w to z before or after agent a_2.

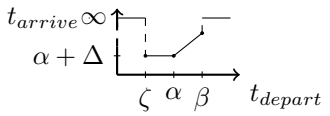

Figure 2: An ATF with parameters ζ, α, β, and Δ.

Any-Start-Time Planning

The original SIPP algorithm is an A^* search with the scalar earliest arrival time at a state as the cost (g-value) of the path to that location. Any-start-time planning (@SIPP) generalizes SIPP to plan for all start times by searching with earliest arrival-time functions (ATFs), rather than scalar g-values (Thomas et al. 2023). This creates the benefit that we keep track of all possible plans, including different start times, arrival times, and paths. An ATF maps the departure time of the agent, when it begins traversing an edge, to the earliest arrival time at the end of the edge. A SIPP graph $\langle S, E, \delta \rangle$ is transformed to an @SIPP graph $\langle S, E, A[e] \rangle$ by generating the arrival time functions $A[e]$ for all edges $e \in E$. For each $e = \langle s_1, s_2, i \rangle$, the safe intervals from the source state s_1 (interval $[t_s^{s_1}, t_e^{s_1}\rangle$), the destination state s_2 (interval $[t_s^{s_2}, t_e^{s_2}\rangle$), and the edge interval $i = [t_s^e, t_e^e\rangle$ are compiled into one ATF. This ATF $A[e]$ is defined from the following four parameters:

$$\zeta = t_s^{s_1} \tag{1}$$

$$\alpha = \max(t_s^e, t_s^{s_1}, t_s^{s_2} - \delta(s_1, s_2)) \tag{2}$$

$$\beta = \min(t_e^e, t_e^{s_1}, t_e^{s_2} - \delta(s_1, s_2)) \tag{3}$$

$$\Delta = \delta(s_1, s_2), \tag{4}$$

where ζ is the earliest time the agent can safely wait at the starting state s_1, α is the earliest time it is safe for the agent to depart state s_1, β is the time at which the edge becomes unsafe to start traversing, and Δ is the time it takes the agent to traverse the edge (Thomas et al. 2023). The ATF $A[e]$ is

then a piecewise linear function constructed by $\zeta, \alpha, \beta, \Delta$:

$$A[e](t) = \begin{cases} \infty & t < \zeta \\ \alpha + \Delta & \zeta \leq t < \min(\alpha, \beta) \\ t + \Delta & \alpha \leq t < \beta \\ \infty & \beta \leq t. \end{cases} \quad (5)$$

Figure 2 visualizes such an arrival time function. Equation 5 considers the ATF of an edge $A[e]$, which can be combined into an ATF for an entire path $A[p]$ (Thomas et al. 2023). Moreover, the $\min$ in the second case is because $\beta < \alpha$ is possible whenever the plan of a path p requires more waiting than feasible before the initial state becomes unsafe. The duration Δ in the edge ATF $A[e]$ is simply the duration δ of e; however, for a path ATF $A[p]$, Δ is the entire path's duration.

Multi-Agent Execution Delay Replanning

We build on the work of Hanou et al. (2024), which applied Any-Start-Time Planning to solve a subset of multi-agent delay replanning problems. They defined the Multi-Agent Execution Delay Replanning (MAEDeR) problem, which is a single-agent problem in the multi-agent setting. A MAEDeR problem is defined by the tuple $\langle N, \mathcal{A}, \mathcal{C} \rangle$. Here, N is the set of components that can be represented as a graph with edges between locations (e.g., a railway network), $\mathcal{A}$ is the set of agents (e.g., trains) navigating through the network, and $\mathcal{C}$ holds the problem characteristics on how agents interact with the network and each other (e.g., speed). A solution to MAEDeR is a function $\mathcal{M}$ that takes an agent $a^{\mathrm{d}} \in \mathcal{A}$ and a positively delayed start time t^{d}, and returns a shortest safe plan for this delayed agent, without affecting the plans of other agents in $\mathcal{A}$; or returns a failure if no such plan is feasible and fuller replanning is required.

Hanou et al. (2024) described two algorithms to solve MAEDeR problems. First, Replanning SIPP performs a SIPP search for the delayed agent, yielding a set of SIPP graphs that can be generated before execution but are searched during execution. Second, @MAEDeR precomputes the SIPP graphs and transforms them into @SIPP graphs (with ATF edges). These are searched before execution using the RePEAT algorithm (Thomas et al. 2023), which repeatedly searches for optimal plans for monotonically increasing start times. The resulting set of plans can be rapidly queried for an optimal plan corresponding to any departure time. Essentially, @MAEDeR replans in advance, so that at execution time, replanning just involves querying the any-start-time plan of the delayed agent. While they find the optimal solution provided only the delayed agent is replanned, the resulting multi-agent plan is not necessarily optimal. Sometimes, a lower-cost overall plan can be found by also replanning other agents.

Comparison The SIPP graph inherently encodes the safe intervals, allowing the delayed agent to choose a time in this interval during which this agent can move between configurations without changing anything regarding the other agents. Any-start-time planning has the additional benefit of precomputing routes for all possible delays for this single agent. @MAEDeR precomputes these any-start-time plans to instantly recover from delays, which can be repeated for multiple delays due to quick updating. However, using the possibility of delaying other agents' plans to provide more or better solutions remains a gap in this line of research.

Flexible Multi-Agent Delay Replanning

Consider a multi-agent path planning problem and a solution to this problem consisting of trajectories for all agents, and assume that the plan of one agent cannot be safely executed (e.g., the agent is delayed); we want to replan that agent, using the temporal flexibility of the other agents. We start with the model of this replanning problem as an @SIPP graph (Thomas et al. 2023). This graph is created from the perspective of the delayed agent, which means that the safe intervals are safe for the delayed agent to not conflict with the movements of all the other agents. Later, we explain how to extend this graph with the temporal flexibility of the other agents. We use the arrival time functions as costs to keep track of different paths to the goal, where some paths may use flexibility and others may not. Formally, a *Flexible Multi-Agent Delay Replanning problem* is defined as the tuple $\langle \mathcal{A}, a^{\mathrm{d}}, N, \delta, t^{\max}, \Pi \rangle$, where

- $\mathcal{A}$ defines a set of k agents,
- $a^{\mathrm{d}} \in \mathcal{A}$ is the *delayed agent*, orignally starting at t_o,
- N is the set of components that can be represented as an @SIPP graph $\langle S, E, A[e] \rangle$ for agent a^{d} where all agents $a \neq a^{\mathrm{d}} \in \mathcal{A}$ are considered dynamic obstacles,
- $\delta(x_1, x_2)$ gives the duration of an edge: the minimum travel time from configuration x_1 to configuration x_2,
- $t^{\max}$ gives the global end time of the scenario,
- Π are safe and valid trajectories; a trajectory $\pi_a \in \Pi$ is a sequence of n_a configuration/time pairs (x_j, t_j) specifying that agent a arrives at configuration x_j at time t_j, with start $x_1 = x_o^a$ and goal $x_{n_a} = x_g^a$, for each agent $a \in \mathcal{A}$.

A trajectory π_a of length n_a is valid if and only if the network N contains every connection between subsequent configurations, and the move respects the minimum travel time: $\forall j < n_a : (x_j, x_{j+1}) \in N$ and $t_j + \delta(x_j, x_{j+1}) \leq t_{j+1}$. The set of trajectories Π is safe if and only if no two agents occupy the same configuration or edge simultaneously. Where a *path* concerns only the physical locations, a *trajectory* is a time-stamped path, the states for an agent's *plan*. A delayed agent may be *rerouted*, changing the path, while other agents can only be *delayed*, keeping the same path but changing the timing. The Flexible Multi-Agent Delay Replanning problem has two use cases with different objectives. The first is to identify the tipping points. The second is to retrieve the optimal solution for a given start time of the delayed agent, where other agents may use their temporal flexibility. While correlated with minimizing makespan, this is not our objective. We define an agent's temporal flexibility as:

Definition 1 (Flexibility)**.** *The* flexibility $f_a(x_j, t_j)$ *of an agent $a \neq a^{\mathrm{d}} \in \mathcal{A}$ in configuration x_j where it arrives at time t_j is the amount of time it can* safely *be delayed in that configuration, while ensuring that 1) all other agents keep their original trajectories, and 2) all agents visit the configurations in the order defined by these original trajectories.*

Algorithm 1 Flexibility for one agent $a \in \mathcal{A}$ based on the original plan Π, where n_a is the length of its plan π_a.

1: $f_a(x^a_g, t_{n_a}) \leftarrow t^{\max} - t_{n_a}$
2: **for** $j \in [n_a - 1, \dots, 1]$ **do** ▷ Configurations visited by plan π_a in reverse order
3: $\quad \tau \leftarrow t_{j+1} - t_j - \delta(x_j, x_{j+1})$ ▷ Local flexibility
4: $\quad m \leftarrow \tau + f_a(x_{j+1}, t_{j+1})$
5: $\quad$ **for** $\overline{a} \in \mathcal{A} \mid a \neq \overline{a}$ **do**
6: $\quad\quad$ **if** agent $\overline{a}$ visits configuration x_j after t_j **then**
7: $\quad\quad\quad t^{\overline{a}} \leftarrow$ earliest time that $\overline{a}$ visits x_j after t_j
8: $\quad\quad\quad m \leftarrow \min(m, t^{\overline{a}} - t_j)$
9: $\quad f_a(x_j, t_j) \leftarrow m$

Algorithm 1 gives a polynomial-time algorithm to compute $f_a(x_j, t_j)$. This is, in fact, equivalent to (backward) propagating the temporal constraints on agent a's configuration x_j at a time t_j as common in Simple Temporal Networks (STNs) (Dechter, Meiri, and Pearl 1991). For each configuration visited by both agent a and some other agent $\overline{a}$ after t_j (line 6), we constrain the flexibility to maintain the current visiting order (line 7-8). An agent's flexibility is determined by the local flexibility (line 3) along its future path, based on fixed visiting times of other agents (line 4). We compute this for all agents except for the delayed agent. When precomputing the any-start-time plan for an agent, that agent is regarded as the delayed agent. Proposition 1 states that the temporal flexibility can always be computed efficiently. Algorithm 1 runs in $O(k \cdot n^2)$ for each agent, where n is the maximum length across all trajectories and k is the number of agents, thus it is polynomial time, proving Proposition 1. The size of the any-start-time plan is linear in the number of agents (each agent can create a new interval), but does not scale with the planning horizon, as intervals are continuous, though runtime does increase linearly.

Proposition 1 (Flexibility)**.** *Given a Flexible Multi-Agent Delay Replanning problem, we can derive the flexibility of all agents in polynomial time.*

Example 1 (Flexibility)**.** *Consider the warehouse example again, replanning delayed agent a_1. We derive the flexibility of agent a_2 along its path to know if we can slightly delay a_2 to let agent a_1 use the middle corridor first. The first part of the path of a_2 is also used by agent a_3, and the flexibility of a_2 cannot influence the path of a_3, so agent a_2 has no flexibility here. However, beyond that part, before reaching the corridor, agent a_2 can wait safely. Configuration u is marked along this section, which agent a_2 originally occupies during $t \in [4, 5\rangle$. We take the scenario to have $t^{\max} = 12$, so agent a_2 originally waits at its goal v during $t \in [9, 12\rangle$, yielding a local flexibility of $f_{a_2}(v, 9) = 3$. For each configuration visited by agent a_2 before its goal, the local flexibility is propagated until the point on a_2's path also used by a_3. The flexibility in the safe point to wait is thus $f_{a_2}(u, 4) = 3$.*

We use the flexibility of all agents to identify a possible reordering between a delayed agent a^{d} and another agent $\overline{a}$ visiting a configuration. The smallest delay at which a reordering improves the objective is called a tipping point:

Definition 2 (Tipping Point)**.** *Take agents a and $\overline{a}$ and their original plans $\pi_a, \pi_{\overline{a}}$, where agent a uses some configuration x° before agent $\overline{a}$ uses this x°. The* tipping point *is the pair (x°, t°), such that until this time t° (not included) the ordering of the agents using this configuration x° is agent a before agent $\overline{a}$, if necessary by delaying $\overline{a}$. After this time t°, the order of the agents is swapped, so $\overline{a}$ uses configuration x° first (at its scheduled time) and agent a waits until the configuration becomes available again.*

Example 2 (Tipping Point)**.** *Agent a_1 needs to replan and can use some of agent a_2's flexibility to still traverse first. We can delay agent a_2 to wait in configuration u during $t \in [4, 8\rangle$ where $t \in [4, 5\rangle$ is its scheduled time, and in the rest of the interval it uses some flexibility $f_{a_2}(u, 4) = 3$. If agent a_1 uses the middle corridor before agent a_2, then it must clear this before $t = 8$, or equivalently, it has to depart its start configuration at the latest by $t = 4$, reaching configuration w at $t = 7$, resulting in a makespan of 12. If agent a_1 reaches w* before *$t = 7$, then agent a_1 can cross the corridor first; otherwise, agent a_2 should move first at its scheduled time, with makespan 12. We thus have a tipping point $(x^\circ, t^\circ) = (w, 7)$ for agents a_1 and a_2.*

FlexSIPP

This section presents the FlexSIPP algorithm, which solves the Flexible Multi-Agent Delay Replanning problem and identifies the tipping points. For any configuration-time pair (x_j, t_j) in an agent's plan π_a, Algorithm 1 gives the flexibility in that configuration, which is a scalar $f_a(x_j, t_j)$. We represent this flexibility in the search by creating an additional safe edge on any edge (x', x_j) in our @SIPP graph for agent a^{d}, so that it can be used for an alternative plan for this delayed agent. This tracks if the flexibility of an agent is used, by updating the configuration-time pairs for the plan of the agent a whose flexibility we have used. Just as with the other @SIPP edges, this edge has an arrival-time function. We construct the *flexible ATF* A^{F} based on an original ATF A of agent a^{d} by changing only the latest start time β to β^{F} to add the flexibility. This ATF A^{F} uses the flexibility of the first agent a^{F} that visits the respective configuration after the delayed agent a^{d} does, unless the flexibility $f_{a^{\mathrm{F}}}(x, t)$ is zero. For every edge e in the @SIPP graph with ATF $A[e]$ for agent a^{d}, we create the $A^{\mathrm{F}}[e, f_{a^{\mathrm{F}}}(x, t)]$ with $\beta^{\mathrm{F}} = \beta + f_{a^{\mathrm{F}}}(x, t)$ based on the original ATF:

$$A^{\mathrm{F}}[e, f_{a^{\mathrm{F}}}(x,t)](t) = \begin{cases} \infty & t < \zeta \\ \alpha + \Delta & \zeta \leq t < \min(\alpha, \beta^{\mathrm{F}}) \\ t + \Delta & \alpha \leq t < \beta^{\mathrm{F}} \\ \infty & \beta^{\mathrm{F}} \leq t. \end{cases} \tag{6}$$

Example 3 (ATF)**.** *In our agent a_1 is replanning a path through the middle corridor (possibly delaying agent a^{d}). We consider the @SIPP edge e_w representing the agent a_1's move from the corridor to configuration w. For this edge's ATF, we do not worry about other conflicts along the path; this is handled by searching for a complete path between safe configurations. Agent a_2 originally occupies configuration w during $t \in [5, 6\rangle$. Agent a_1 can either take this*

edge e_w before a_2, represented by A_1 constructed by $\zeta = \alpha = 0$, $\beta = 3$ (the latest a_1 has to depart to arrive before a_2), and $\Delta = 1$ (just one cell), or after a_2, which is represented by ATF A_2 constructed by $\zeta = 7$ (the earliest it can be here after a_2), $\alpha = 7$, $\beta = t^{\max} - 1 = 11$, and $\Delta = 1$. However, we have another option (besides rerouting through the left corridor) to let agent a_2 wait, using some of its flexibility $f_{a_2}(u, 4) = 3$ (Example 1). In this case, agent a_2 uses w during $t \in [8, 9\rangle$, and agent a_1 thus has a flexible ATF A^{F} constructed by $\zeta = 0$, $\alpha = 3$, and $\beta^{\mathrm{F}} = 3 + 3 = 6$ (derived by updating β from A_1), and $\Delta = 1$.

Search

We find the any-start-time plan for agent a^{d} using the RePEAT algorithm (Thomas et al. 2023) on the @SIPP graph with the additional flexible ATFs. These ATFs define the flexibility that can be used during the search. The only update in the search procedure is the tracking of flexibility. When the delayed agent a^{d} uses a flexible ATF A^{F} to construct its new path, this means that another agent $a^{\mathrm{F}} \neq a^{\mathrm{d}}$ has to use some of its flexibility, which thus impacts that agent's plan. To ensure feasible plans for the other agents, this impact must be accounted for. When the agent a^{F} is delayed in some configuration x by at most $f_{a^{\mathrm{F}}}(x, t)$, every configuration-time pair in the single-agent plan $\pi_{a^{\mathrm{F}}}$ that is visited after configuration x must now be updated. Each such configuration $\overline{x}$ was originally safe during some interval $[\underline{t_s}, \underline{t_e}\rangle$ (before agent a^{F} visits), and also safe during some interval $[\overline{t_s}, \overline{t_e}\rangle$ (after agent a^{F} visits). These safe intervals are then updated to $[\underline{t_s}, \underline{t_e} + f_{a^{\mathrm{F}}}(x, t)\rangle$ and $[\overline{t_s} + f_{a^{\mathrm{F}}}(x, t), \overline{t_e}\rangle$, respectively.

Example 4 (Search with flexibility)**.** *We show how the arrival time functions change during search when using flexibility. In this case, agent $a^{\mathrm{F}} = a_2$ and $a^{\mathrm{d}} = a_1$. We have flexibility $f_{a_2}(u, 4) = 3$ and take $t^{\max} = 12$. The path ATF ${A_1}^{\mathrm{F}}[p]$ is constructed by $\zeta = 0, \alpha = 1$, $\beta^{\mathrm{F}} = 4$ and $\Delta = 4$, because it has to clear the corridor before agent a_2 uses it (before $t = 1$ no flexibility is used). Agent a_2 has a path ATF $A_2[p]$ constructed by $\zeta = 0$, $\alpha = 3$, $\beta = 1$, and $\Delta = 9$, which means it has to depart its start configuration before $\beta = 1$ (because agent a_3 is right behind), but has flexibility along its path to wait at most $\alpha = 3$, while the plan without waiting takes $\Delta = 9$. Assume agent a_1 uses the corridor first, starting at $t = 3$, forcing agent a_2 to wait until $t = 7$ to enter the corridor from configuration u, taking a delay of 2 by using some of its flexibility. If we consider the safe intervals on configuration w, which were originally safe during $[\underline{0}, \underline{4}\rangle$ (before a_2 visits as planned) and $[\overline{6}, \overline{t^{\max}}\rangle$ (after a_2 visits as planned), these are updated to $[\underline{0}, \underline{6}\rangle$ and $[\overline{8}, \overline{t^{\max}}\rangle$, accounting only for a_2's moves as we are replanning a_1 and agent a_3 does not use the corridor.*

Example 5 (Any-start-time plan)**.** *Figure 3 shows the results from FlexSIPP for the running example. We see that FlexSIPP finds four paths for agent a_1: 1) A_1 lets agent a_1 depart until $t = 1$ without further delays; 2) ${A_1}^{\mathrm{F}}$ forces agent a_2 to use its flexibility; 3) the path reroutes through the left corrirdor, but after $t = 5$ it is equally fast to wait for agent a_2, which is preferred as the path stays the same; and 4) a_1 can only depart after a_2 has cleared the corridor (during $t \in [5, 7\rangle$ arrival time does not change), and must depart before $t = 8$ to its goal in time ($t^{\max} = 12$). This shows the tipping point at $t = 4$ for starting time (see Example 2). We also see the paths found by @MAEDeR, which reroute during $t \in [1, 4\rangle$ because no flexibility is used.*

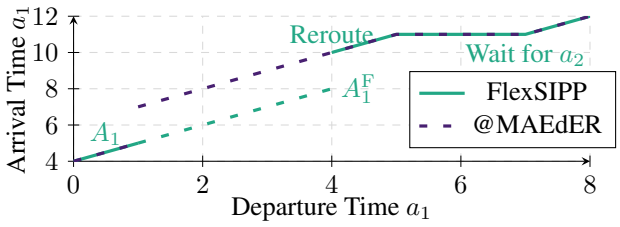

Figure 3: Results on warehouse example from Figure 1.

Using the search procedure described above, we now show that we find the best replanning solution given the agent's flexibility in polynomial time (Proposition 2). The flexibility is derived in polynomial time (Proposition 1), and the construction of the @SIPP graph is also done in polynomial time. The original @SIPP graph $\langle S, E, A[e]\rangle$ based on the network N and set of agent $\mathcal{A}$ is of order $O(|N| \cdot |\mathcal{A}|)$ (Hanou et al. 2024). FlexSIPP adds more safe intervals, but at most one per edge in the original @SIPP graph, so the @SIPP graph is still in the order $O(|N| \cdot |\mathcal{A}|)$. Each path is split into simple paths with one origin and destination, so we can have several paths for one agent, and the problem size thus scales linearly with the number of simple paths. The branching factor in the A^* search is bound by the number of agents; so, it runs polynomially in the number of agents.

Proposition 2. *If a replanning solution exists that involves only delaying one of the other agents according to their flexibility (Definition 1) and possibly rerouting the delayed agent, then we find this solution in polynomial time.*

Tipping Points FlexSIPP returns an any-start-time plan for agent a^{d} and the tipping points with other agents, meeting the first objective of Flexible Multi-Agent Delay Replanning. During search, we track the flexibility used of any agent $a^{\mathrm{F}} \neq a^{\mathrm{d}}$. For each plan found in the any-start-time plan, with a path ATF $A^{\mathrm{F}}[p]$ using flexibility, the tipping point is then the β^{F} parameter, which is the latest time that the agent a^{d} can still safely use the configuration, based on the flexibility of a^{F}. For the entire path of a_1, we have $\beta^{\mathrm{F}} = 4$ (Example 4), which ensures that both agents a_2 and a_1 still arrive before $t^{\max} = 12$. So, we see indeed the tipping point identified in Example 2, placed on the starting time of agent a_1. The location of the tipping point is the first point along the delayed agent's path where the other agent passes, which is configuration w. To find the tipping point time stamp, we take the $\beta + \Delta$ of that ATF. For the ATF A_1 of a_1 moving to w, we had $\beta^{\mathrm{F}} = 6$ and $\Delta = 1$ (Example 3), resulting also in the tipping point $(x^\circ, t^\circ) = (w, 7)$.

Minimize Total Delay FlexSIPP can also be used in the second use case of Flexible Multi-Agent Delay Replanning:

retrieving the optimal solution for a given start time of the delayed agent. We compute the total delay by comparing each agent's arrival time at its goal with its originally scheduled arrival time. Given the any-start-time plan, we can derive both the fastest plan for the delayed agent and the plan with the least total delay over all agents.

Rescheduling For each plan in the resulting any-start-time plan, FlexSIPP determines the agent(s) whose flexibility is used (if any at all), and if so, when and where the waiting time of the other agent(s) must be adjusted. These agents can only be forced to wait, not to change their path, and this never causes conflicts, because only feasible delays (i.e., not influencing another agent) are used as input to the search. The any-start-time plan with flexibility is precomputed before execution, so plan updates occur in constant time and are, in fact, almost instantaneous. After the update, FlexSIPP can be used to recompute any-start-time plans for any agent, or to simply query the first plan if a new delay is encountered before the any-start-time plan is recomputed. The safe intervals are continuous, so upon a delay, the exact time point of the delay can be queried for the correct safe interval to start the new plan from. FlexSIPP searches over the @SIPP graph, where each edge is thus an ATF representing a continuous time span for safe traversal between configurations. The tipping points are single time points, which are retrieved from the end of the safe traversal (β parameter of the ATF).

Case Study: Rotterdam to Schiphol

To evaluate the effect of using the flexibility of other agents, we consider the example of rerouting trains in a busy railway network. When disturbances occur in railway network operations, trains are delayed, and in some cases, they must be rerouted. If the tracks are already occupied close to full capacity, it is very difficult to fit this rerouting into the schedule. It thus becomes necessary to adjust the schedule slightly to accommodate delayed trains. Specifically, we consider the tight schedule in the Netherlands, where train services are frequent and expected to become even more metro-like in the next few years (ProRail 2024)[p.21]. The railway network in the Netherlands is mostly double-track, allowing trains to be rerouted to an alternative track. Then, replanning must consider the trains originally scheduled to use that track, preferably without disrupting their service too much.

A high-speed line (HSL) runs between Rotterdam Central and Schiphol Airport to facilitate easy travel between two of the largest cities: Amsterdam and Rotterdam. However, the HSL is particularly prone to issues: train punctuality is only 69% on this line, compared to 89% on the rest of the network (NS 2024a). When the HSL is unavailable due to large disruptions, trains scheduled to use it must be rerouted over the regular network, through Leiden and The Hague (Figure 1 in the Appendix[1]). In this case, the previously discussed algorithm (@MAEDeR) for the Multi-Agent Execution Delay Replanning problem falls short because the timetable is so tight that no complete path fits between the regular trains. So, we must slightly delay some of the regular trains to allow for the HSL-rescheduled trains to be scheduled at all.

In practice, railway companies (in the Netherlands) use *train handling documents*, which can be understood as a flow chart saying what the appropriate action is if a train traveling a specific path is delayed (NS 2024b). These documents thus specify the tipping points, when to delay the other trains, and when the delayed train should allow other trains to proceed first. For example, in the case of the Rotterdam-Leiden-Schiphol line, the tipping point used in industry for regular intercity and sprinter trains is 7 minutes; regular trains on the path between The Hague and Leiden can thus be delayed by 7 minutes to allow another train to proceed first (Kemmeren 2025)[p.14]. Currently, these documents are created manually, although they are not specified for every situation (e.g., this Eurostar case). So, FlexSIPP provides a safe and automated alternative.

Evaluation

We evaluate our method on the case study and compare it to @MAEDeR, which does not allow delaying other agents.

Q1 Can FlexSIPP provide tipping points for replanning a high-speed train in the Netherlands?

Q2 Does the any-start-time plan found by FlexSIPP hold more information than the one returned by @MAEDeR

Q3 How does FlexSIPP's runtime compare to @MAEDeR?

Q4 Can FlexSIPP plan with sequential delays?

Experimental Setup

FlexSIPP is implemented in two stages: generation of the @SIPP graph in Python (implemented for both railways and MAPF instances), and search in C++ that takes an @SIPP graph and computes the any-start-time plan for the delayed agent with the specified amount of flexibility. FlexSIPP is released as a Python package (Hanou et al. 2026).[2] The railway experiments were run on an AMD Ryzen 5600X with 32GB of RAM and a search time limit of 1800s, and the MAPF experiments on an M1 chip with 16 GB RAM.

Case Study: Eurostar Rerouting

We reroute the Eurostar train on the Rotterdam-Schiphol line via Leiden. The case study uses confidential infrastructure data provided by the Dutch railway company ProRail. The Appendix reports on the constructed network N from this data, details of the flexibility computation, and an analysis of the runtime components. We used four scenarios based on real-world events; the details are also reported in the Appendix. For each scenario, we ran FlexSIPP to identify the tipping points for each train, as planned for the Eurostar. Table 1 shows the tipping points found for one of the four scenarios. Each row represents a regular train that is delayed by the time specified in the 'Delay' column to accommodate the new Eurostar train passing at the point indicated by 'Location'. The trains are numbered by their index in the scenario, and the delay location provides the city where the train

[1] See extended version at https://arxiv.org/abs/2601.04884.

[2] Figure 4 is generated with v2.0 the rest of the figures with v1.0.

Train	Location	Delay	Tipping point
64	Leiden	00:00	03:59
32	Hoofddorp	02:07	05:00
20	The Hague	01:39	01:16
	Delft	00:00	05:00
51	The Hague	00:00	05:00
16	Leiden	00:00	05:00
11	Hoofddorp	03:42	04:58
70	Schiedam	00:00	05:00
18	Delft	00:00	05:00

Table 1: Tipping points found by FlexSIPP for Scenario 4 (82 trains over a 474-minute timespan). Times in `mm:ss`.

must be delayed. As the railway industry does not define any tipping points for this Eurostar scenario, we cannot make a direct comparison. However, we see that in most cases, the other train incurs no delay to accommodate the new Eurostar, and we have several options to accommodate the Eurostar. So, for **Q1**: yes, FlexSIPP provides tipping points for real-world cases with many trains in a congested network.

We also compared the number of paths in the any-start-time plan found by @MAEDeR and FlexSIPP, as shown in Table 2. Here, the important point is that several paths can be combined into a single path. FlexSIPP identified significantly more paths than @MAEDeR. However, upon careful examination of the plans found, those devised by @MAEDeR force the Eurostar to wait until all other trains have passed, and then it can move at the end of the scenario. Our algorithm uses $t^{\max}$, which is set to some time after the last train arrives at its destination, therefore allowing @MAEDeR to plan trains at the end. This obviously results in a large overall delay, while FlexSIPP can plan the Eurostar earlier, reducing the overall delay and answering **Q2**.

MovingAI Benchmarks

FlexSIPP is most useful in contexts where paths are constrained, but agents have some buffer time along them. We already demonstrated in the case study and now also show that FlexSIPP works on the regular MovingAI MAPF benchmarks (Stern et al. 2019). Therefore, we chose two specific maps with all corridors of width 1: the maze of 128x128 with scenarios of 20 and 50 agents, and the warehouse of 20x40x10 with scenarios of 50 agents. We computed the agents' original plans using Priority-Based Search (Ma et al. 2019). For the maze, this yielded 25 and 22 scenarios for 20 and 50 agents, respectively, and 10 for the warehouse.

Scenario	FlexSIPP	@MAEDeR
1	5487	279
2	4418	564
3	4845	340
4	525	150

Table 2: Number of paths found to accommodate Eurostar.

			Diff	**Time**		**Path Found**	
Map	k	`f`		F	M	F	M
maze	20	0	-7.32	25.70	24.29	69%	65%
		3	-13.21	38.05	18.43	65%	57%
		5	-14.88	7.33	4.79	66%	56%
		8	-24.36	18.84	29.44	78%	62%
	50	0	-22.77	95.54	89.34	74%	63%
		3	-23.68	101.24	93.05	68%	53%
		5	-44.33	27.87	17.08	75%	60%
		8	-33.66	34.09	29.68	81%	60%
ware	50	0	0.05	13.36	12.40	66%	66%
		3	0.00	78.04	21.55	63%	63%
		5	-0.36	17.85	16.24	80%	80%
		8	0.01	13.27	12.18	66%	66%

Table 3: Total delay comparison for maze scenarios with 20 and 50 (k) agents and warehouse (ware) scenarios with 100 agents, where one agent is delayed, and extra flexibility (`f`) is added in the initial paths. Shows the average total delay difference (@MAEDeR - FlexSIPP) and the percentage of runs when a path was found, along with the average search time over those for FlexSIPP (F) and @MAEDeR (M).

First, we considered the optimal path for the delayed agent with a delayed start time, based on the total delay of all agents, computed by summing the arrival time minus the original arrival time before the delay was handled. To represent robustness and evaluate the effect of flexibility, we modified the plans while maintaining the paths of the original plan, so that whenever two agents use the same node within `f`timesteps, the latter agent waits at the previous node until there is a difference of `f`timesteps, where $\texttt{f} = 0$ is the original plan from Priority-Based Search. For each scenario, we randomly delayed one agent by selecting a location along its path and sampling a delay between 0 and the time the next agent enters that location. We repeated this three times per scenario to obtain different random delays. The path returned by FlexSIPP was compared to the path returned by @MAEDeR, where the latter cannot delay other agents, whereas the former is optimal given that other agents can only be delayed, not rerouted. Table 3 shows the total delay for all agents that was found by FlexSIPP and @MAEDeR, respectively, and FlexSIPP was able to find paths in many more cases for the maze. The difference (Diff) is averaged across scenarios and the three iterations per scenario; a negative value indicates that FlexSIPP's plan resulted in less overall delay. For the warehouse, FlexSIPP does not outperform @MAEDeR because the agents interact much more frequently in their paths in this map, so they have less flexibility. While FlexSIPP was a bit slower than @MAEDeR, we see that with additional flexibility in the original plans, FlexSIPP was able to find a better overall plan than @MAEDeR, answering **Q3**. We also note that sometimes neither algorithm could find a new plan for the delayed agent, and full replanning for all agents would be required.

Finally, we used the maze map with one scenario of

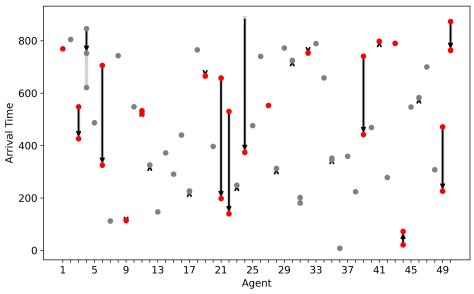

Figure 4: Total delay progression over sequential updates. The input delay is the agents' cumulative imposed delay, showing the delays in resulting arrival times. $\times$ marks runs where no path was found (and @SIPP graph not updated).

50 agents and their original paths ($\texttt{f} = 0$). Then, over 25 sequential iterations, we randomly delayed one agent. FlexSIPP returned the fastest path for this delayed agent, after which we updated our @SIPP graph and handled the next delay. Figure 4 shows the difference in delays, where we take the input delay from the randomized delays and show the difference in original arrival times and resulting arrival times after each delay of FlexSIPP compared to @MAEDeR. FlexSIPP always results in fewer delays and sometimes even yields a lower total delay than the input: if the original plan was not optimal or if a previously delayed agent is rerouted and a better plan becomes available. To answer **Q4**: FlexSIPP can indeed plan with sequential delays.

Related Work

The flexibility computation in Algorithm 1 is very similar to vertex slack (Jiang, Lin, and Li 2025), although we define flexibility as the maximum amount we can delay an agent at a configuration without delaying *any* other agent, whereas vertex slack is the maximum amount an agent can be delayed at a configuration without delaying another *specific* agent. Other order-switching algorithms were introduced before. The Switchable (Jiang, Lin, and Li 2025) and Bidirectional (Su, Veerapaneni, and Li 2024) Temporal Plan Graphs define order switches between agents, repairing a plan by introducing delays. Kottinger et al. (2024) proposed to let an agent wait along its path, and the Location Dependency Graph (Liu et al. 2024) avoids conflicts and deadlocks by coordinating the input wait and move actions. However, neither method allows the originally delayed agent to be rerouted. Moreover, we provide a stronger guarantee on the impact (i.e., delay) of the original delay on the other agents. While the (Switchable) Action Dependency Graph (Hönig et al. 2019; Berndt et al. 2020) also seems related, it reconsiders the plans and orderings of all agents. In contrast, FlexSIPP focuses solely on replanning a single delayed agent when needed. Contingency plans (Nekvinda, Barták, and Kalech 2021), where the delayed agent can choose an alternative path, also precompute alternative routes, and collision-free paths are guaranteed, as only one agent can choose an alternative path. Conversely, FlexSIPP reuses the precomputed @SIPP graphs, allowing it to quickly handle a new delay by updating the safe intervals of the newly delayed agent.

FlexSIPP relates to replanning for the Multi-Agent Pathfinding (MAPF) problem, in which agents must reach their goal locations without conflicts (Stern et al. 2019). Švancara et al. (2019) used an extended problem formulation that allows new agents to appear and replans a different number of agents under various guarantees. Two proposed methods do not allow changes to other agents' plans, as in MAEDeR's problem formulation, whereas the others impose fewer restrictions. FlexSIPP strikes a new balance between computational efficiency and optimality by keeping the paths of all other agents but adjusting their delays to create an opening for the new agent to plan. While we focus on recovering from a delay (using precomputation), Atzmon et al. (2018) proposed k-robust planning, providing plans that are robust to delays of upto k timesteps.

Our SIPP representation enables continuous-time solutions, related to Continuous MAPF approaches, such as Continuous Conflict-Based Search (CCBS) (Andreychuk et al. 2019), although it focuses on planning and does not allow replanning or flexibility. In Lifelong MAPF, new goals are continuously added to the current scenario during execution. So, the plan must be continually updated to reflect the current situation. While different strategies for replanning are used, such as replanning all agents, or just the affected agents (Zhang et al. 2024), these methods do not reuse the knowledge from solving the scenario in earlier timesteps. FlexSIPP could also be applied here to more efficiently use the information at different points in time and find a better overall solution. FlexSIPP replans a single agent, which could iteratively build a multi-agent plan by adding one agent at a time, closely related to Prioritised Planning (Morag et al. 2025). However, deciding on the agents' priorities would also affect their flexibility.

Conclusion

FlexSIPP uses the available temporal flexibility in a multi-agent plan to precompute the any-start-time plans for all agents, so that upon a single-agent delay, this agent can immediately retrieve its safe paths, which may delay other agents, without propagating this delay further. After such a delay, FlexSIPP handles new delays sequentially by updating the precomputed @SIPP graphs and returning the best solution for the single delayed start time. The flexibility ensures no new conflicts arise with the delay, thus not further impacting any other agents. So, we can reorder a delayed agent with other agents from the original plan. We demonstrated our method on a common MAPF benchmark and showed that it is effective in real-world settings, transforming detailed railway infrastructure into a simplified model via safe-interval path planning. While able to handle the continuous-time component and different agent speed profiles, our method is also effective in simpler grid-world settings, such as Multi-Agent Pathfinding. Our method brings the single-agent replanning approach using any-start-time planning closer to a true multi-agent pathfinding approach.

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/.

References

Andreychuk, A.; Yakovlev, K.; Atzmon, D.; and Stern, R. 2019. Multi-Agent Pathfinding with Continuous Time. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, volume 28, 39–45. Main Track.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Bartá, R.; and Zhou, N.-F. 2018. Robust Multi-Agent Path Finding. In *Eleventh Annual Symposium on Combinatorial Search*, volume 9.

Berndt, A.; Van Duijkeren, N.; Palmieri, L.; and Keviczky, T. 2020. A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph. *ICAPS 2020 DMAP workshop*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence*, 49(1-3): 61–95.

Goverde, R. M.; Corman, F.; and D'Ariano, A. 2013. Railway line capacity consumption of different railway signalling systems under scheduled and disturbed conditions. *Journal of Rail Transport Planning and Management*, 3(3): 78–94.

Hanou, I.; Kemmeren, E.; Thomas, D. W.; and de Weerdt, M. 2026. FlexSIPP Code and Data for "Precomputing Multi-Agent Path Replanning Using Temporal Flexibility".

Hanou, I.; Thomas, D. W.; Ruml, W.; and de Weerdt, M. 2024. Replanning in Advance for Instant Delay Recovery in Multi-Agent Applications: Rerouting Trains in a Railway Hub. In *International Conference on Automated Planning and Scheduling*, volume 34.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics and Automation Letters*, 4(2): 1125–1131.

Jiang, H.; Lin, M.; and Li, J. 2025. Speedup Techniques for Switchable Temporal Plan Graph Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22): 23212–23221.

Kemmeren, E. 2025. *Introducing Flexibility in Any-Start-Time Safe Interval Path Planning: A Case Study on the Dutch Railway Network*. MSc thesis, Delft University of Technology.

Kottinger, J.; Geft, T.; Almagor, S.; Salzman, O.; and Lahijanian, M. 2024. Introducing Delays in Multi Agent Path Finding. *Proceedings of the International Symposium on Combinatorial Search*, 17(1): 37–45.

Liu, Y.; Tang, X.; Cai, W.; and Li, J. 2024. Multi-Agent Path Execution with Uncertainty. *Proceedings of the International Symposium on Combinatorial Search*, 17(1): 64–72.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 7643–7650.

Morag, J.; Zhang, Y.; Koyfman, D.; Chen, Z.; Felner, A.; Harabor, D.; and Stern, R. 2025. Prioritised Planning: Completeness, Optimality, and Complexity. *Journal of Artificial Intelligence Research*, 84.

Nekvinda, M.; Barták, R.; and Kalech, M. 2021. Contingent Planning for Robust Multi-Agent Path Finding. *Proceedings of the International Symposium on Combinatorial Search*, 12(1): 185–187.

NS. 2024a. Annual Report: Our Concession Indicators.

NS. 2024b. Het komt maar weinig voor, maar soms wordt een station overgeslagen. https://nieuws.ns.nl/het-komt-maar-weinig-voor-maar-soms-wordt-een-station-overgeslagen/.

Pachl, J. 2021. *Railway Signalling Principles*. Technische Universität Braunschweig.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation*, 5628–5635. IEEE.

ProRail. 2022. Netverklaring. Technical report, ProRail.

ProRail. 2024. Jaarverslag. Technical report, ProRail.

Stern, R.; Sturtevant, N. R.; Felner, A.; Keonig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; Boyarski, E.; and Barták, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS 2019: The 12th Annual Symposium on Combinatorial Search*, volume 12. Position Paper.

Su, Y.; Veerapaneni, R.; and Li, J. 2024. Bidirectional Temporal Plan Graph: Enabling Switchable Passing Orders for More Efficient Multi-Agent Path Finding Plan Execution. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16): 17559–17566.

Thomas, D. W.; Shimony, S. E.; Ruml, W.; Karpas, E.; Shperberg, S. S.; and Coles, A. 2023. Any-Start-Time Planning for SIPP. In *ICAPS23 Workshop on Heuristics and Search for Domain-Independent Planning*.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online Multi-Agent Pathfinding. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, volume 33. AAAI Technical Track: Planning, Routing, and Scheduling.

Zhang, Y.; Chen, Z.; Harabor, D.; Le Bodic, P.; and Stuckey, P. J. 2024. Planning and Execution in Multi-Agent Path Finding: Models and Algorithms. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, volume 34, 707–715.

Appendix: Railway Replanning with Flexibility

Figure 5 shows the railway network in the Netherlands, with the section line between Schiphol and Rotterdam highlighted.

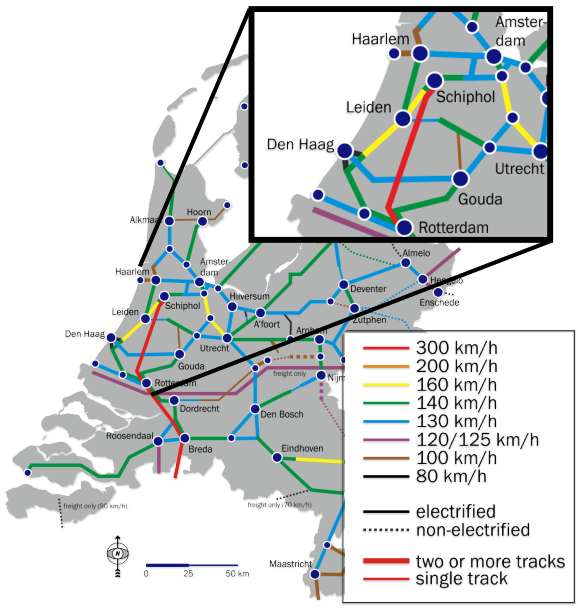

Figure 5: Railway Network in the Netherlands showing the maximum speed on each segment. Zoom-in of the area between Rotterdam and Amsterdam, both the regular track over Leiden in yellow, as well as the high-speed section between Rotterdam and Schiphol in red (ProRail 2022)[p.199].

The railway network is divided into *blocks*, and only one train is allowed inside a block at a time. A common safety procedure uses a trackside signal to inform train drivers whether a block is occupied. To calculate when a block is free to enter, the railway industry uses *blocking times*, which can be calculated precisely using the length of the block, the speed of the train, the reaction time of the driver, and the time it takes the train to clear the block (Goverde, Corman, and D'Ariano 2013). The blocking times can be visualized over the travel distance as in Figure 6. In this example, two trains follow each other on the same route. As long as the blocking times do not overlap, no conflict arises between these two trains. We find the minimum separation of two trains, called the *headway*, when two trains directly follow each other in at least one block, which is referred to as the critical block. The flexibility of trains is thus easily visualized as the gaps between their blocking times, in industry referred to as *buffer time*, which also depends on the flexibility in the next block.

Now, we show how to model the Railway Replanning Problem as a Flexible Multi-Agent Delay Replanning problem. Each block is a configuration x, allowing one agent to enter a block at a time. We represent the railway network in a routing graph, connecting blocks based on whether a train travels directly from one to another (Kemmeren 2025). The blocking times are used to calculate the traversal times Δ between blocks. From a given timetable, we generate the @SIPP graph for the delayed train and extract the trajectories of other trains, which are the dynamic obstacles. Based on this timetable, we then compute the flexibility of the other trains, given by the blocking time computations. Finally, we also allow the delayed agent to be a completely new agent that was not included in the original plan Π. In this case, the @SIPP graph that is given as input for Flexible Multi-Agent Delay Replanning has safe intervals that avoid conflicts with all agents from the original plan.

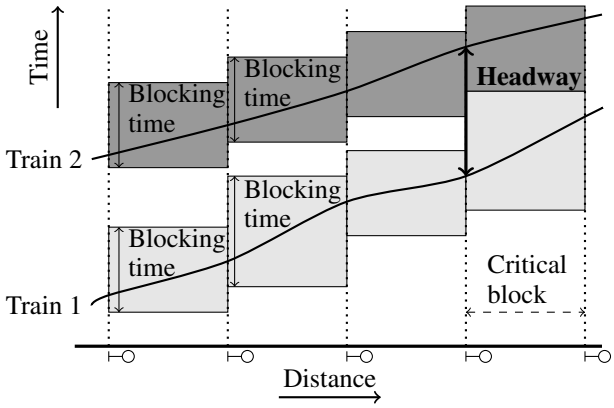

Figure 6: Two blocking time staircases of two trains along the same track. The rightmost block is the critical block. Adapted from Pachl (2021).

As a heuristic in the search procedure, we use the shortest distance to the goal configuration, without considering any other agents. As this never overestimates the cost of reaching this destination, the heuristic is *admissible*. The heuristic is also *consistent*, as it always decreases by at most the minimum edge duration as we move closer to the goal location of the agent.

Experiments

From the infrastructure, the resulting routing graph of the entire Dutch railway network consists of 9700 nodes with 247,600 routes, averaging 24.5 outgoing routes per node, which is reduced in size to 2.48 outgoing nodes by discarding shunting yards (Kemmeren 2025)[p.35]. For the case study experiments, we use the timetable data provided by the Dutch Railways.[3] We gathered the data on July 8, 2025, constructing four scenarios: 64 trains over a 432-minute timespan (resulting in 12293 safe edge intervals), 82 trains over a 456-minute timespan (resulting in 16510 safe edge intervals), 77 trains over a 492-minute timespan (resulting in 14356 safe edge intervals), and 82 trains over a 474-minute timespan (resulting in 16989 safe edge intervals), respectively.

[3] https://www.ns.nl/reisinformatie/ns-api

Table 4: Table reporting the components of the runtime in seconds of the two algorithms for the different scenarios.

Time Components	Scen 1	Scen 2	Scen 3	Scen 4
Creation Routes N	72.2	72.1	72.1	72.1
Conflict gen.	284.8	254.0	240.8	215.1
Interval gen.	10.7	9.4	8.7	8.9
Flexibility gen.	1.1	0.9	0.9	0.9
Search FlexSIPP	1190.8	1005.6	964.7	249.5
Search @MAEDeR	68.8	220.7	94.9	22.0

Runtime Results

Table 4 shows the runtime components for the four scenarios, comparing FlexSIPP and @MAEDeR. The 'Creation routes N' generates the routing graph from the railway infrastructure, then unsafe intervals (the conflicts) are generated, including the blocking time calculations, which are converted to safe intervals, upon which the flexibility is computed. The last two rows give the search time of the different algorithms, as the first steps are the same for all three algorithms. FlexSIPP spends more time searching than @MAEDeR. This is because it finds many more paths and thus needs to restart the search more often. The other time components are equal across both methods, as they use the same construction of the @SIPP graph from the underlying data. As the scenarios use the whole Dutch railway network as an underlying graph, this takes quite some time. However, the computations done by FlexSIPP can also be precomputed, similar to @MAEDeR. So, while FlexSIPP does require more runtime to search through the graph, as this is significantly larger, this is reasonable when we can precompute this for the entire railway network in the Netherlands.