Technische Universiteit Delft

Master of Science Thesis in Computer Science

Introducing flexibility in any-start-time safe interval path planning:
a case study on the Dutch railway network

Eric Kemmeren

Supervisors:
Prof. Dr. Mathijs de Weerdt
Ir. Issa Hanou

September 2, 2025

Delft University of Technology

Introducing flexibility in any-start-time safe interval path planning: a case study on the Dutch railway network

Master's Thesis in Computer Science

Algorithmics group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Eric Kemmeren

September 2, 2025

Author
Eric Kemmeren

Title
Introducing flexibility in any-start-time safe interval path planning: a case study on the Dutch railway network

MSc presentation
September 9, 2025

Graduation Committee

Prof. Dr. M. M. De Weerdt	Delft University of Technology (chair)
Ir. I. K. Hanou	Delft University of Technology
Prof. Dr. R. M. P. Goverde	Delft University of Technology

Abstract

During the daily operation of the railway network, ProRail is responsible for handling delays and planning ad hoc train movements. Train handling documents aid the traffic controllers in common situations. But when multiple trains are delayed, and these documents do not apply, they are left to their own expertise.

In this thesis, we introduce FlexSIPP, an algorithm to plan or replan agents in an existing multi-agent plan. FlexSIPP builds upon the prior works of any-start-time safe interval path planning, where the current routes of the agents are seen as moving obstacles. FlexSIPP loosens this restriction by introducing flexibility: the ability for an agent to delay its plan while minimally impacting other agents.

This algorithm is evaluated on the Dutch railway network. By finding tipping points, that is, the moment it is better to switch the order of two trains on the track to minimize the delay, we can recreate train handling documents. We show that FlexSIPP finds the same solutions within a minute in the case that no other trains are delayed. This implies that FlexSIPP is also able to aid traffic controllers in the case that other trains are delayed.

Preface

With my academic journey at the TU Delft comming to an end, I can look back at a transformative six years long chapter of my life. With many widely varying challenges during these years I was able to grow on a personal and professional level. I would like to thank everyone that was part of that.

Designing and developing an algorithm has challenged me greatly in this thesis project. Thats why I would like to first of all thank my supervising professor, Mathijs de Weerdt, for the sparring sessions and continous support. Even as my explanations were not as clear as they could be. With the fast responses to my emails and critical questions I was able to push this thesis further.

Furthermore, I would like to express my gratitude to my daily supervisor Issa Hanou for guiding me through this project. The weekly meetings helped me keep on track and maintain perspective on the bigger picture of the project.

I would also like to thank my contact at ProRail, Wilco Tielman, for the meetings and supplying me with information about the workings of the railway network in the Netherlands.

Lastly, I am deeply thankfull for family and friends throughout the duration of my study and this project. Especially for supporting me during the moments I too busy to do anything else than work on some project and sleep.

Eric Kemmeren

Delft, The Netherlands
September 2, 2025

Contents

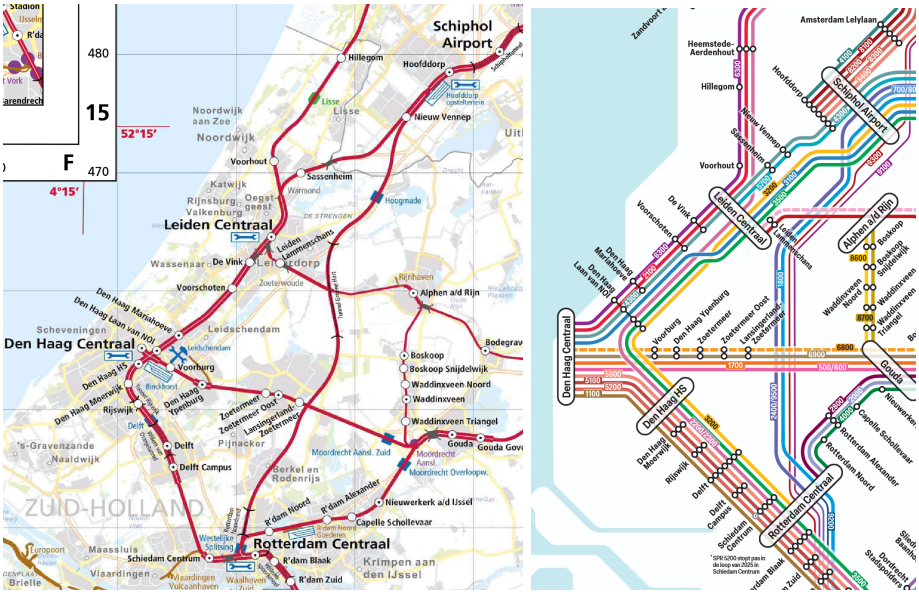
Chapter 1

Introduction

In the Netherlands, ProRail is responsible for the daily operation of the railway network. These responsibilities include handling delays and planning new train movements. Local traffic controllers handle these operations between intersections in the railway network. Network traffic controllers coordinate the traffic over large areas (Hansen and Pachl, 2008). The traffic controllers are of utmost importance for the full utilization and safety of the railway network. To aid these traffic controllers in common situations, there are train handling documents, called TADs (*Trein Afhandelings Documenten in Dutch*). These documents specify what actions to take in case of delays and incidents. These actions can include, but are not limited to, delaying other trains, skipping a station, or canceling a train. For some situations, these TADs specify when two trains should swap order on the track. This is to reduce the oil spill effect that a delayed train can cause, which causes an increasing number of trains to be delayed.

The downside of these documents is that they are only usable for specific problems, and do not combine well. Applying multiple of these turnkey solutions can be infeasible, as they are not made to fit together. To improve the support of the traffic controllers, we need to look for an approach that takes into account the current state of the network. When a train is delayed, it can cause the following train to want to occupy the same part of the track at the same time. This is called a conflict. We aim to solve this conflict by supplying the traffic controller with up-to-date information to re-plan the delayed train for the specific current state of the network, instead of a TAD.

The problem of finding routes for all trains in the railway network can be seen as a multi-agent pathfinding problem, where each train is an agent traversing the network. Multi-agent pathfinding is the problem of finding routes for a group of agents from starting locations to goal locations. Finding these routes can be computationally expensive (Yu and LaValle, 2013). After creating the multi-agent plan, we arrive at the execution phase. During the execution, agents can be forced to deviate from the plan. Reasoning can



(a) ProRail railway map. (ProRail, 2024) (b) NS line network. (NS, 2025)

Figure 1.1: Two depictions of the railway network between Schiphol and Rotterdam

be that an action takes longer than expected, or a sudden delay occurs. This is due to the uncertainties of the problem. Cacchiani et al. (2014) describe disturbances in our problem of train routing as "relatively small" perturbations. For example, a train leaves the station later due to the doors not closing properly. Larger perturbations are described as disruptions and can include tracks becoming unavailable because of an accident. These disruptions have a large impact on the routes that trains can take.

One solution for creating a multi-agent plan that can handle small delays is k-robust planning. In the initial planning phase, it is taken into account that an agent can be delayed by up to k timesteps while not invalidating the plan (Atzmon et al., 2018; Chen et al., 2021). This does sacrifice some optimality for robustness and is not guaranteed to handle delays greater than k. For these delays, replanning of all agents is required.

Another approach is finding a new path for the delayed agent using the existing multi-agent plan. This creates a set of problems called multi-agent execution delay replanning (MAEDeR) (Hanou et al., 2024). Focusing on the delayed agent, it is possible to reduce the multi-agent pathfinding problem to a single-agent problem by treating other agents as moving obstacles. This allows for the initial planning to be optimal and allows delayed agents to be replanned efficiently, although not necessarily attaining a global optimum when a delay occurs. The solution proposed in that line of work uses a form of safe interval path planning (SIPP). This algorithm identifies time

windows in which a route is available. As an example, take the high-speed railway line between Schiphol and Rotterdam on the Dutch railway network shown in Figure 1.1a. When a disruption occurs on this line, there are two options for the trains that are on this route. Wait for the disruption to clear, or take the longer and busier route through Leiden and The Hague. Depending on the departure time of the train and when the disruption will be cleared, either one of the options can be the best. SIPP encodes the time window that the route is available efficiently, such that it can apply heuristic search on it. To model the MAEDeR problem, the other agents are seen as immovable objects, just like the disruption. Because the planning for these agents is known, the routes can be marked as unsafe when an agent occupies the space. Replanning of an agent happens around these unsafe routes. This method does not take into account the flexibility of other agents. Trains that normally take the route through Leiden and The Hague can deviate from their planned timetable slightly without impacting other trains or even their own departure time at the next station, as the time spent at that station can be longer than necessary (ProRail VenD VACO, 2016). Delaying these agents at strategic points allows us to plan a new agent between existing agents.

In the execution of SIPP, it is expected that the starting time is known. The search algorithm can easily identify the fastest route to a point. It assumes that when a new route is found that is slower, that route does not need to be taken into account. @SIPP (Thomas et al., 2023) removes this assumption. It will calculate for every possible departure time in the future what the fastest route is. If during the search a point is already visited, @SIPP will still use this point if the route to that point is no longer available due to a later starting time. It still, however, assumes the obstacles it is planning around are immovable.

In this thesis, we provide an extension to @SIPP that allows us to remove this assumption and delay the plan of other agents. With this extension to @SIPP, called FlexSIPP, we take a step closer to finding the optimal plan for the multi-agent pathfinding problem, from a single-agent perspective. With this we can also identify tipping points, like the TADs specified, that tell us when it is best to swap the order of two agents. To determine the validity of the proposed method, a study will be performed on the Dutch railway network.

FlexSIPP will be able to delay the plan of other agents by identifying flexibility in the other agents' plans. The flexibility can fall under two categories: buffer time and recovery time. Buffer time is the amount of time an agent can delay its plan while not conflicting with other trains. Recovery time is the time delta between the scheduled running time of its route and the absolute minimum time needed. A delayed agent can decrease its delay by this amount.

To summarize, in this thesis, we will be answering the following questions:

1. Does including flexibility in any-start-time safe interval path planning allow us to find a plan that improves the total cost in a multi-agent plan?
2. Can FlexSIPP identify tipping points when trains need to swap positions to minimize the total delay?
3. Is FlexSIPP able to aid traffic controllers in real-time?

Chapter 2

Background

Before introducing flexibility in any-start-time safe interval path planning, we first introduce (any-start-time) safe interval path planning. To conduct the case study on the Dutch railway network, we discuss how signaling works on the railway, and what work has already been completed to model the railways as a graph that can be used with any-start-time safe interval path planning. Lastly, we introduce train handling documents that aid traffic controllers when conflicts occur.

2.1 Path Planning

Safe Interval Path Planning (SIPP) (Phillips and Likhachev, 2011) is a way to navigate around dynamic obstacles. The dynamic obstacles can be any obstruction of available paths during some part of the search. That can entail other agents moving around or the closure of a path. Due to these obstacles, not all paths are always available.

As trains are restricted in their movements to the tracks, we can model the railway network as a directed graph, with individual nodes and edges for each direction. Each state in this graph corresponds to a location on the track. In SIPP, the edges correspond to a safe interval between two locations in the graph. As an example, we can use the two routes between Schiphol and Rotterdam. When a disruption occurs on the high-speed line, this route is not available. Only after the disruption is cleared would an edge be added between Schiphol and Rotterdam that uses the high-speed line. The detour through The Hague and Leiden would correspond with a safe interval that is available, but slower than the high-speed line.

Formally, we can define a SIPP problem as a tuple $\langle G, \delta, x_o, x_g \rangle$, where the graph G consists of states S and edges E. The state represents a node in the graph with a location and safe interval, and is defined as $s = \langle x, i \rangle \in S$. The edges represent when it is possible to go from the origin state to the destination state with a unique interval, represented by the tuple

$e = \langle u, v, i \rangle \in E$. The variable x denotes a location, and the locations x_o and x_g are the origin and goal locations, respectively. For every state and edge, there is an $i = \langle t_s, t_e \rangle$ that denotes the interval from t_s to t_e where the corresponding state or edge is safe. An edge has a nonnegative cost of $\delta(u, v)$, implying that an agent taking an edge from u to v at time t, will arrive at $t + \delta(u, v)$. To optimally solve the SIPP problem, we need to find a path from x_o to x_g through the graph G, minimizing the total cost over the path. It is also possible to wait in a state s, until the end of the safe interval associated with that state is reached. This allows for the reduction that it is always better to be in a state earlier, as any later time can be reached by waiting. SIPP then only tracks the earliest arrival time in a state, and with A^* search finds the optimal route departing immediately.

In Any-start-time SIPP (@SIPP) (Thomas et al., 2023), SIPP is augmented to allow for an unknown start time t_0. It achieves this using arrival time functions (ATFs). These arrival time functions specify the arrival time for a given departure time. Now, it is not always better to only track the earliest arrival time at a state, as search algorithms like A^* do. It also depends on when the route that arrives at a location is available. Because of the uncertainty in t_0, the earliest arrival time at a state can become unreachable when starting later. For instance, the train that is deciding to wait for the disruption to clear on the high-speed line is occupying a platform at Schiphol. Another train will soon be arriving at that same platform, so it needs to start moving before the other train arrives.

In @SIPP, the ATFs are defined as an edge with four parameters: ζ, α, β and Δ, where ζ denotes the time the agent can start waiting at the origin state u. This origin state has the safe interval $i^u = \langle t_s^u, t_e^u \rangle$. The destination node v has a safe interval of $i^v = \langle t_s^v, t_e^v \rangle$. The variable α is the earliest possible time the agent can start traversing the edge $e = \langle u, v, i^e \rangle$, with a safe interval $i^e = \langle t_s^e, t_e^e \rangle$. To traverse the edge, the agent must be able to safely depart the starting location, avoid conflicts when traveling, and arrive at the end of the edge. The variable β is then the latest time the agent can depart from the origin state, and lastly Δ is the duration to traverse the edge e. Equations 2.1 show the calculations for these four variables.

$$\zeta = t_s^u \tag{2.1a}$$
$$\alpha = max(t_s^u, t_s^e, t_s^v - \delta(u, v)) \tag{2.1b}$$
$$\beta = min(t_e^u, t_e^e, t_e^v - \delta(u, v)) \tag{2.1c}$$
$$\Delta = \delta(u, v) \tag{2.1d}$$

Using these variables, we can define the ATF:

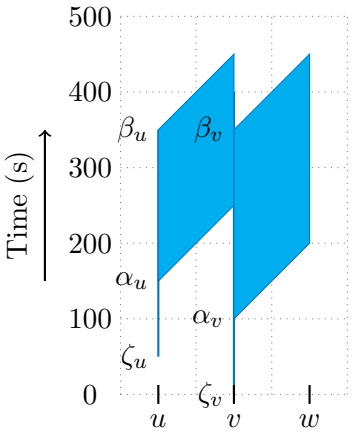
$$f[\zeta_e, \alpha_e, \beta_e, \Delta_e](t) = \begin{cases} \infty & t < \zeta_e \\ \alpha_e + \Delta_e & \zeta_e \leq t \leq min(\alpha_e, \beta_e) \\ t + \Delta_e & \alpha_e \leq t \leq \beta_e \\ \infty & \beta_e < t \end{cases} \tag{2.2}$$

As it is impossible to depart when the edge is not safe, we set an infinite arrival time for when departing before ζ or after β. If the train departs Schiphol before α and β, then we need to wait before we can safely depart at time α, and then the train still needs to traverse the edge. If the train departs any later than this, it also arrives later if it is following the same route.

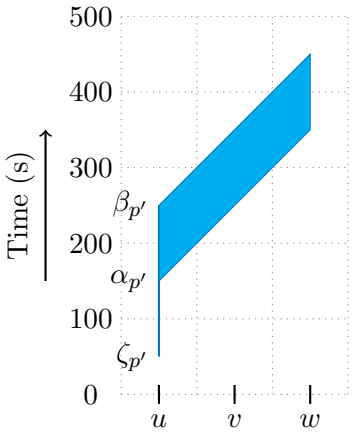
An ATF is defined for every initial edge in the search space. When constructing a path, it is possible to combine the ATFs of the edges of the path to compose the ATF for the path. The ATF is piecewise linear, which entails that in the intervals defined in Equation 2.2, the arrival time only increases linearly. This allows for multiple ATFs to be efficiently combined. As all edges have a piecewise linear ATF, the path ATF will also be piecewise linear. To start the construction of the path ATF, we first use the insight that a path of a single edge will result in the path ATF and edge ATF being equal. If we have a path p to some state that has an outgoing edge e, we can combine the ATF of the path p and edge e into the ATF of p'. We need to account for the four variables of the ATF when combining two ATFs. The variable $\zeta_{p'}$ is equal to ζ_p, as the earliest time the start of the path is safe will not change when adding an edge to the path. What can change is the amount of waiting the agent must and can do along the path. Forcing the agent to wait longer on the path will increase α, whereas limiting the amount of time an agent can wait will decrease β. The duration $\Delta_{p'}$ is simply the sum of the duration of the path and the edge. Formally, these insights give us the recursive definition of the equivalent edge.

$$\zeta_{p'} = \zeta_p \tag{2.3a}$$
$$\alpha_{p'} = max(\alpha_p, \alpha_e - \Delta_p) \tag{2.3b}$$
$$\beta_{p'} = min(\beta_p, \beta_e - \Delta_p) \tag{2.3c}$$
$$\Delta_{p'} = \Delta_p + \Delta_e \tag{2.3d}$$

An example where two ATFs are combined is shown in Figure 2.1. Initially, the first ATF of a path is equal to the outgoing ATF at the current node, thus when going from u to v, $f_p = f_u$ in the example. When combining f_p and f_v, the variables α_v and β_v will be offset by the time to reach v, hence the term Δ_p in Equations 2.3b and 2.3c. The time the calculated path is available is the time between $\zeta_{p'}$ and $\beta_{p'}$. This duration can only decrease when expanding the path further. Only when $\beta_{p'}$ becomes smaller than $\zeta_{p'}$,



(a) Arrival time functions of the two edges $\langle u, v\rangle$ and $\langle v, w\rangle$. Δ is the slope from the starting node to the destination node.



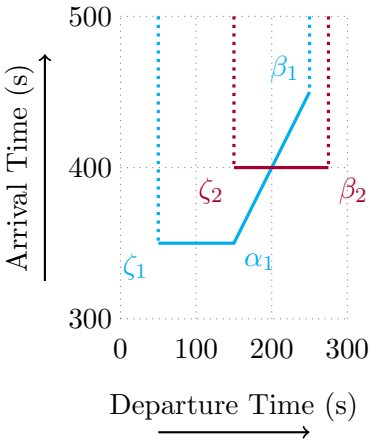
(b) Resulting arrival time function of p' from u to w when combining the ATFs from Figure 2.1a.

Figure 2.1: Example of applying the recursive definition of the equivalent edge to calculate the arrival time function from u to w by combining the arrival time function encountered on the path.

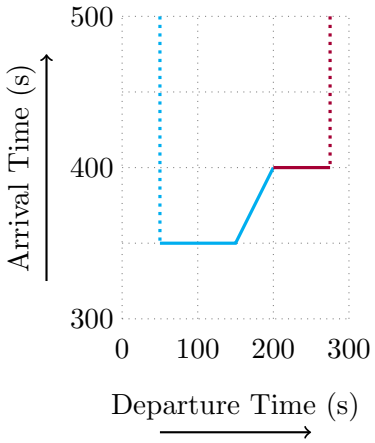
the path becomes unavailable. It can be that $\beta_{p'}$ becomes smaller than $\alpha_{p'}$, in this instance the agent must depart from the origin between $\zeta_{p'}$ and $\beta_{p'}$, and will always arrive at the same time at the destination, at $\alpha_{p'} + \Delta_{p'}$.

Using A* search, the optimal solution can be found. Every time a node is expanded during the search, the variables are calculated as above. When a path is found, A* search is repeated to search all possibilities of departing from the origin node. The reasoning is that we want to calculate a path for all possible starting times, and a new path could be found from the time that the initially found path departs from the origin. The search ends when the agent can not depart from the origin before it becomes unsafe. The difference with SIPP and @SIPP is that the former will find the fastest path immediately, whereas the latter will find the fastest path departing starting at t_0, till the starting state becomes unavailable.

After finding every path from x_o to x_g, we can combine the arrival time functions by exploiting their piecewise linearity. In Thomas et al. (2023), every combination of combining two arrival time functions is explored. In this thesis, we explain the concept using one example as shown in Figure 2.2. The idea is to combine every ATF into a compound ATF (CATF), such that we minimize the arrival time for every departure time, where the agent can take multiple paths depending on the departure time. In the example in Figure 2.2a, we have the arrival time functions from Figure 2.1b, and we have also found another path available between $\zeta_2 = 150s$ and $\beta_2 = 275s$.



(a) Arrival time functions of two paths p_1 and p_2, where p_1 corresponds with the ATF in Figure 2.1b, and p_2 always arrives at $400s$



(b) Combination of the two arrival time functions.

Figure 2.2: Combination of two arrival time functions, resulting in a compound arrival time function (CATF).

From $200s$ onwards, it becomes faster to use this path. In Figure 2.2b, we find the resulting compound arrival time function with an extra break point at the point where the original ATFs intersect.

2.2 Railway Infrastructure

"The main railway infrastructure is fitted with signalling systems, safety and communication systems for the safe and controlled handling of train traffic." (ProRail, 2024). Most systems work by dividing the railway network into blocks, where only one train is allowed inside a block. To inform the drivers, ATB and ETCS Level 1 use trackside signals, whereas ETCS Level 2 uses an interface inside the driver cabin. The maximum allowed speed at that time is also indicated to the driver through these signals.

Explanation and modeling of the railway infrastructure is extensively discussed by Hansen and Pachl (2008). Due to the large braking distances of a train, it is impossible for the train to suddenly stop at a red signal when going full speed. Distant signals indicate to the driver that a red signal is coming up such so the speed can be reduced, allowing the train to stop at the red signal. Using distant signals is called one-block signalling. Two-block signalling is when the main signal can also indicate if the next signal is red. In two-block signaling, it is impossible for the next block to suddenly have a red signal, as it needs to be indicated first. This leads us to conclude that when a train is traveling in a block, and has passed a green distant

signal, other trains can not enter the block our train is going to enter, as the distant signal would need to indicate that. The distance between the signals is called the approach distance, the approach time of the block is the running time of a train over the approach distance. This is called the approach time of the block. In the industry, it is said that the train has a reservation on this block.

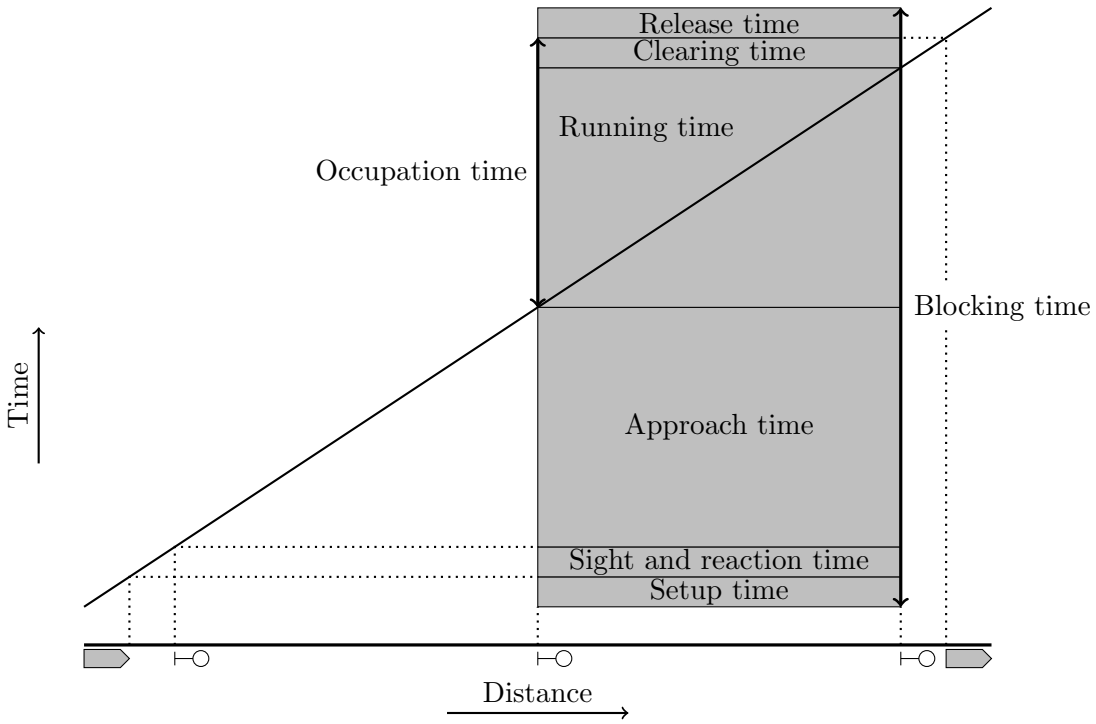


Figure 2.3: Distance time diagram of the blocking time of a train in two-block signalling adapted from Goverde et al. (2013).

To calculate when the next train can enter a block, we can calculate the blocking time of a train. The blocking time of a block is the time that an agent reserves a block or is inside of the block. Figure 2.3 shows the blocking time. It takes into account when a train driver can first see the signal, including how fast they can react. During the time the train is in front of the block we are calculating the blocking time for, it is called the approach time. The occupation time is when a part of the train is inside the block. Lastly, the release time is the time it takes for the signal to update. The size of the blocking time depends on the length of the block and the train speed. Hansen and Pachl (2008) also discuss how to calculate the running time in a block when a train changes speed throughout the sections.

When signals are close together, as can happen in highly utilized sections

in multi-aspect signalling, the braking distance may be larger than the length of the block. In such cases, a train will reserve multiple blocks in front. The approach time will then include all blocks from when the train passes the first signal that it needs to start braking to come to a complete stop.

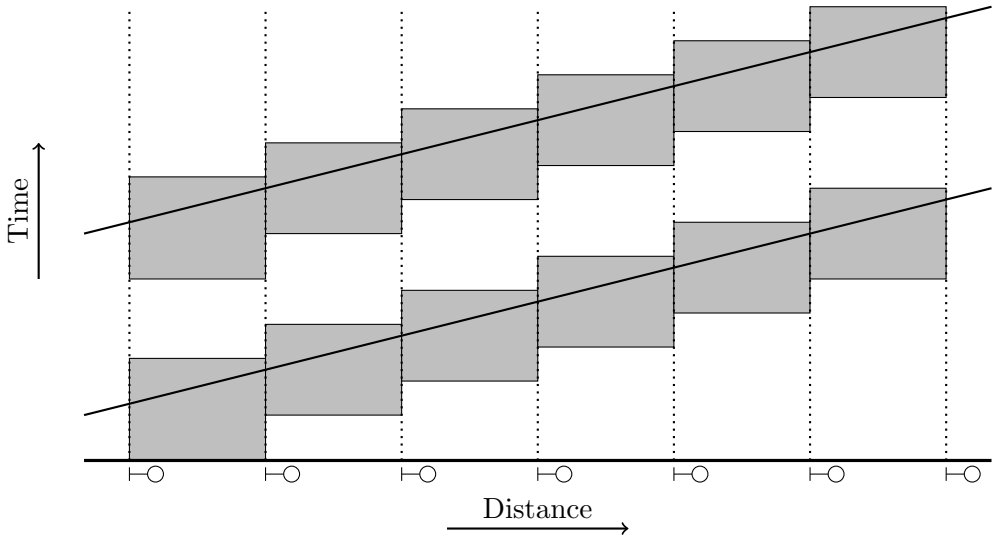
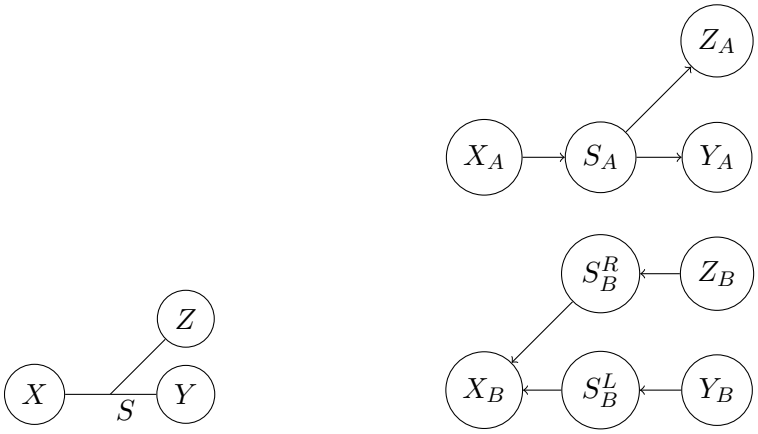


Figure 2.4: Blocking times of blocks on a track adjacent to each other, called a blocking time stairway. Two trains are simulated. Adapted from Pachl (2020).

Figure 2.4 shows the blocking time on a section of a track with two trains. Using these blocking times, we can calculate the minimum amount of separation that is needed on this line for the two trains not to interact with each other. We achieve this by compressing the blocking staircase diagram and moving the blocking times of the trains as close together as possible, such that they never create a conflict. This minimum separation between the points is called the line headway. This headway is different for trains following each other and trains going in the opposite direction, but it can also be different between different types of trains.

2.3 Reduction from a railway track to @SIPP

To solve the routing of a train in a multi-agent execution delay replanning problem at stations, Hanou et al. (2024) proposed a method to construct a graph from the railway track to run @SIPP on. This reduction starts with the insight that a train can not turn around suddenly. This leads us to have two nodes for a single part of the track. One node is the A side, where trains can only go in one direction, and the other node is the B side,



(a) A standard switch S in a track, a train can not move directly from z to y.

(b) Graph of a standard switch, agents can only move in the direction of the arrows.

Figure 2.5: Topology of a switch and corresponding track graph

going in the opposite direction. These nodes are only connected at tracks where turning around is allowed, for example, at a station. At a switch, this directed graph generates more nodes. With the assumption that this is happening at a railway yard, trains travel slowly enough to run 'on sight'. A train can then wait right in front of the switch when another train passes. For purposes of @SIPP, we generate a new unique node for every location it can wait. The result is a Left and Right node of the switch, as shown in Figure 2.5b. In the A direction, there is no L and R node as there is no extra place for a train to wait. In the B direction, it can wait before the switch coming from Y for a train coming from Z.

When a train can turn around on a piece of track, the A and B sides will be connected. Now that it is possible to generate a graph from the railway infrastructure, we need to determine what the safe intervals are for SIPP. For this we will first show how to generate unsafe intervals that can be inverted to create safe intervals and the ATF as used in @SIPP.

Unsafe intervals are closely related to blocking times, but use a default headway for the train separation. This paper uses the headway as the minimum amount of time after a train has passed a location till the next train reaches that location. This is different from the headway as explained in Section 2.2, where it includes the running time of the leading train. This leads to that when a train travels through a node, that node is unsafe for a new train to enter. Only when the train has fully passed the node and the headway time has passed can a new train enter. Take $t_s^{X_A}$ as the time an agent a enters node X. This node has an unsafe interval i_{X_A} starting from $t_s^{X_A}$ and stopping when the train has fully passed the starting point,

plus the headway following ϵ_f. The time it takes for a train to fully pass the starting point is determined by the train length $\lambda(a)$ over the velocity of that train $\nu(X_A, a)$ at the current node. The resulting equation is equal to

$$i_{X_A} = \langle t_s^{X_A}, t_s^{X_A} + \frac{\lambda(a)}{\nu(X_A, a)} + \epsilon_f \rangle \tag{2.4}$$

Node X_b also has an unsafe interval, as it resembles the same piece of track but in the opposite direction of X_a. This results in using the crossing headway instead of the following headway, other parts are equal.

$$i_{X_B} = \langle t_s^{X_A}, t_s^{X_A} + \frac{\lambda(a)}{\nu(X_A, a)} + \epsilon_c \rangle \tag{2.5}$$

When a train turns around, the driver has to walk to the other side of the train and start up again. The duration of walking the train to the other side of the train uses the driver's walking speed ω and train length $\lambda(a)$. Again, for spacing purposes, a new train can only enter this track after the headway time has passed. This results in the following unsafe interval when turning around:

$$i_{Z_B} = \langle t_s^{Z_A}, t_s^{Z_A} + \frac{\lambda(a)}{\omega} + \epsilon_f \rangle \tag{2.6}$$

The interval for the opposing node Z_B is exactly equal.

For a switch, consider Figure 2.5. When a train travels from X to Y, it passes through node S_a. For this node, the unsafe interval can be calculated with Equation 2.4. For the two opposite nodes S_b^R and S_b^L, only the node that corresponds with the direction the train is going will have an unsafe interval, as in Equation 2.5. In this case, that is node S_b^L, as a train can wait at S_b^R. In the B direction, the node S_a always receives the unsafe interval for the opposite direction as in Equation 2.5.

With all the unsafe intervals that can be generated for all other agents in the multi-agent delay replanning problem, it is possible to invert these intervals to create safe intervals for the planning agent to use. These same intervals can be used to create the ATFs with the same approach as Thomas et al. (2023).

2.4 Train handling documents

In the Netherlands, the traffic controllers from ProRail coordinate the movements of the trains. During normal operations, a system called ARI (Automatische Rijweginstelling in Dutch) sets all the switches and signals in the correct state, following the timetable. When a delay occurs, the traffic controllers step in and handle the situation. To aid the traffic controllers, ProRail and the NS have created documents that specify what actions to take when certain delays occur. An example of order changes of trains on a

Trainseries	From	Decision point	Vtgranges	Handling_trainseries
3500o	Ledn	Gv	$7 \rightarrow$	4300o,3500o
600o	Zl	Hde	$5 - 10$	700o, 600o, 6100o, 9000o
600o	Zl	Hde	$11 - 15$	700o, 6100o, 600o, 9000o
600o	Zl	Hde	$16 \rightarrow$	700o, 6100o, 9000o, 600o
3000e	Sgn	Amr	$7 \rightarrow$	3000o, 3000e

Table 2.1: Example of a train handling document, if a train from the series "Trainseries" is delayed by "Vtgranges" minutes at the location "Decision point", it needs to follow the order of trains as defined in "Handling_trainseries" at the location "From".

track is shown in Table 2.1. To better illustrate how to use such a document, take the first row of the table. It specifies a rule for a train for the series 3500*o*, that is, a train between Rotterdam and Schiphol, and the green line in Figure 1.1b. The "o" implies that we look at the odd direction, from The Hague to Schiphol. If a train from this series has a delay between 7 and 10 minutes, or more than 11 minutes at the station The Hague HS with abbreviation Gv, it will swap the order with the train from the series 4300*o* at the station Leiden Centraal. This train is the Sprinter starting at The Hague Central, to Schiphol Airport as shown with the teal colored line in Figure 1.1b.

For this TAD, we only show train order changes. A TAD can also specify short turning or skipping stations. Early turning is when a train does not finish the whole route to its destination, but turns around at an earlier station and starts its journey back. Using these simple rules helps the traffic controllers to efficiently handle delays in a standardized way. In certain situations, multiple TAD rules may be used at the same time. This does require more thorough planning from the traffic controller to make sure no conflict occurs. In this thesis, we will show a method to create these train handling documents dynamically, taking into account the current situation on the network. This has the advantage that if multiple TAD rules need to be used, traffic controllers can be assured that the decisions do not create conflicts and apply to the current network occupation.

Chapter 3

Related Work

This thesis focuses on advising traffic controllers for conflict resolution. When a conflict occurs, we are interested in finding a valid route for all trains involved and getting back to the original planned schedule. This requires replanning of a part of the schedule. In this chapter, we discuss the current state of the art multi-agent pathfinding (MAPF) methods for replanning agents. We first delve into the MAPF literature, where the focus is on creating an efficient and fast algorithm that produces good results. Then we discuss literature that is more focused on the real-world train network, solving the planning of all trains using mixed integer linear programming.

3.1 Multi-Agent Pathfinding

The multi-agent pathfinding problem is defined by having multiple agents that need to reach their goals without creating conflicts with each other (Stern et al., 2019). Švancara et al. (2019) have extended the MAPF problem by introducing the ability of new agents to appear in the problem. By introducing these agents, the existing plan is modified to create a plan for these agents. In their paper, four methods to solve this online MAPF problem are discussed. The first method is by replanning the new agent around existing agents. This is similar to the approach of (any-start-time) safe interval path planning as discussed in detail in Section 2.1. The second method determines the path for multiple new agents at once, while still avoiding all other agents. Neither of these methods can change the plans of other agents. Using them may thus lead to solutions of poor quality. The third method simply replans all agents in the network. This can give the optimal plan, but requires longer computation time. It also does not consider the already computed ongoing plan. The novel approach of this paper is Online Independence Detection. This method routes an agent, ignoring every existing agent in the network. When a conflict occurs, both agents are replanned. This is repeated until a conflict-free path is found.

Although no exact measurements of replanning a single agent were given, it was used as a baseline for every measurement, as it was "done extremely quickly and it is always good to have some solution rather than no solution". It was found that for densely populated grids, replanning all agents gave a ten to twenty-two percent improvement in the cost of all paths. For sparsely populated grids, the improvement was only around one percent. The single-agent replanning algorithm uses the existing agents as moving obstacles and plans around them, creating a conflict-free path. By introducing flexibility in all agents in the single-agent replanning algorithm, it is possible to change the delay these moving obstacles. This leads to an unexplored area of research where we are finding a multi-agent path in a single-agent environment.

3.2 Mixed Integer Linear Programming

There are many different approaches to rescheduling in Railway Networks (Fang et al., 2015). A common method is Mixed Integer Linear Programming (MILP). These models optimize an objective function. Examples of such an objective function are to minimize the sum of all delays (Rudan et al., 2013) or to maximize the robustness of the network. Maximizing robustness entails finding a timetable in which delays to trains are less likely to propagate and impact the scheduled times of other trains (Fischetti et al., 2009).

Small disturbances can be absorbed by a stable and robust timetable (Goverde, 2007). For larger delays, one operation that can be applied is the reordering of trains. Van den Boom et al. (2011) proposed an MILP method that optimally reschedules trains to minimize the total delay. They constrain the model to only allow reorderings at stations where sufficient infrastructure capacity is available. Later work of Kersbergen et al. (2016) builds upon this model by extending the set of constraints on the model to more accurately represent the railway network. In this model, reorderings of trains are controlled by decision variables. Each combination of two trains on a track has one control variable that determines the order. Current operations use TADs to determine the order of two trains on a track. In these TADs, the amount of delay required to swap positions is given. The current literature only determines the order on track for a given situation. To create TADs, we also need information on the amount of delay when it is better to swap order.

To plan a new train in the railway network, it is important to have an accurate overview of the current position of the existing trains. Wang et al. (2025) show the impact of the headway on different driving styles to calculate a train path envelope. These train path envelopes inscribe the train's conflict-free trajectory (Quaglietta et al., 2016). As long as the train stays

within this envelope, it will not conflict with other trains and will be on time for its arrival at the next station.

Zhou (2023) has developed an MILP model that first focuses on finding a rescheduled plan that is close to the original schedule, and then adds flexibility in this plan. In this research, the flexibility was used to create a robust schedule that can handle subsequent delays. This flexibility contains three components: departure, early arrival, and late arrival flexibility. This flexibility is closely related to the train path envelopes. The maximum amount of departure flexibility is equal to the Departure Tolerance Response driving strategy as described by Wang et al. (2025), as both depart later from the station while still being on time at the next stopping point. While this flexibility is important to create a robust schedule, when the delay of a train increases such that it does not fit in the train envelope, the schedule becomes invalid. In this case, a new schedule must be made.

In recent works, Versluis et al. (2024) worked out an MILP model that uses ETCS L2 with trackside train detection. It can take into account different speed profiles of trains in the network. It uses more advanced signalling systems such as distance-to-go. This system has signals inside the driver's cabin that allow the appropriate signal to be shown when it is needed to the driver, based on the braking distance of the specific train it is driving. This allows for the headway to be reduced between trains. Later work looks at ETCS L2 with onboard train integrity monitoring (Versluis et al., 2025). Onboard train integrity monitoring allows for smaller block sizes, as small as twenty meters. This decreases the headway that trains have on a straight part of the track. ETCS L2 with trackside train detection is only used on a few railways in the Netherlands, with plans to convert the whole railway network slowly. For this thesis, we will be focusing on the NS'54 signalling system, as we are performing a case study on the current state of the network.

The MILP models discussed all have one factor in common: they replan all agents, with the constraint that agents are not allowed to depart before their original departure time. In the MAPF literature, it is shown that this method is generally slower than single-agent replanning, as it is a much harder problem (Nebel, 2024).

Chapter 4

Method

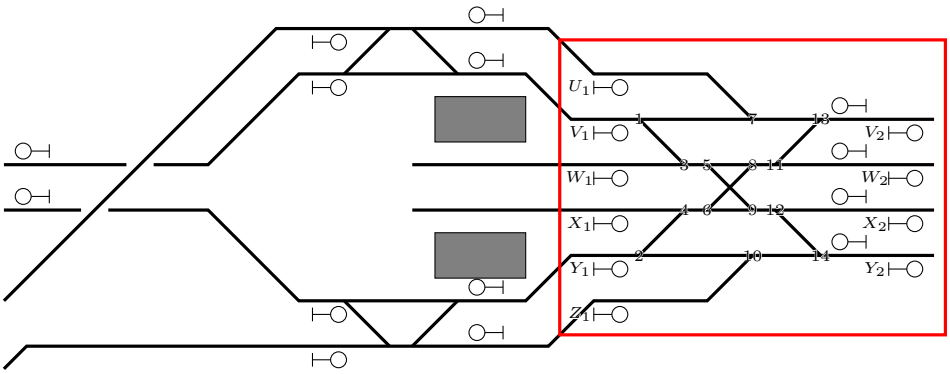
To be able to aid traffic controllers and determine tipping points on the railway network, we need a close-to-realistic representation of the railway network in a directed graph. We first introduce this routing graph, which is based on the track graph by Hanou et al. (2024). Then we determine the unsafe intervals in the routing graph, using the blocking times theory. In Section 4.3, we introduce flexibility into @SIPP. This flexibility allows us to find more feasible paths in the multi-agent path-finding problem. With this, we introduce how to determine the amount of flexibility of a train. Lastly, in Section 4.4 we show how to use this flexibility by introducing FlexSIPP.

4.1 Creating the Routing Graph

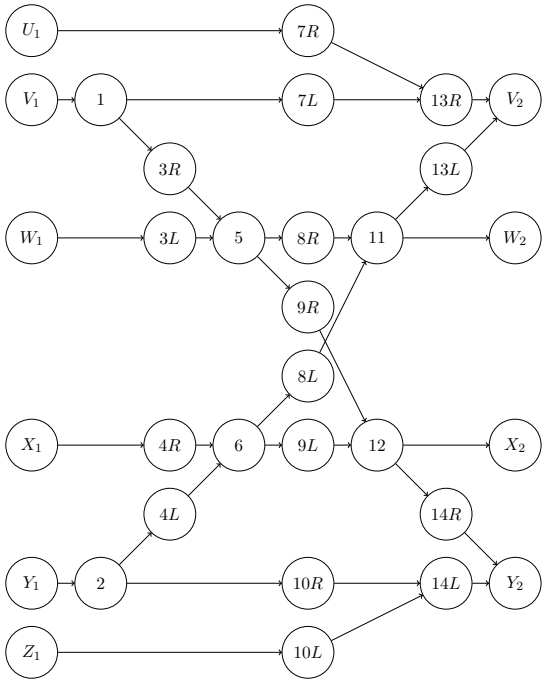
In the model of the railway network of Hanou et al. (2024), a node in the graph represents a physical point on the track. To move to model blocks in the railway network, a node should represent the location where a block starts. This is accompanied by a physical signal at this location. The edges in this new graph correspond with all possible routes the train can take to the start of the next block. In Figure 4.1, the railway network of Hoofddorp is shown, together with the track graph and route graph. Hoofddorp is a station on the route between Schiphol and Rotterdam. At this station, the high-speed line in the direction of Rotterdam splits off with one track for each direction. In the direction of Leiden, there is also one track in each direction. During disruptions on the high-speed line, this becomes the first area of interest, as trains leaving normally on two tracks now need to share one.

The routing graph is again a directed graph. To create this graph, we can traverse the track graph, starting at every node in the track graph that is located at the start of the block. By traversing the track graph, storing what track parts we traverse till we encounter the start of the next block, denoted by a signal in the same direction as we are traveling, we can create

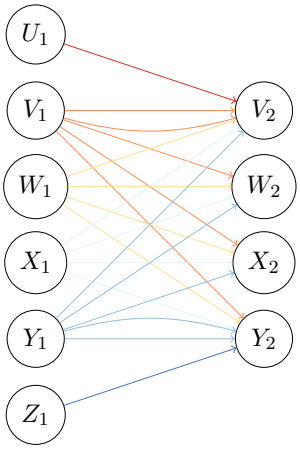
Figure 4.1: Graphs showing the reduction from the railway network to the route graph at Hoofddorp.



(a) Railway network at Hoofddorp, the red box specifies the area that is expanded further in a track and route graph.



(b) Track graph of a part of Figure 4.1. This graph is limited to only showing one direction of travel, moving from the left to the right



(c) Route graph of signals of Figure 4.1 showing routes from signal to signal.

the edges in the routing graph. Every edge in the routing graph will have track parts associated with it that the train will travel over if that route is taken. When two routes share some nodes in the track graph, both routes will become unavailable if a train takes one of the routes, for the time the train is in the shared part of the route.

4.2 Blocking Times in a Routing Graph

To calculate the unsafe interval for the blocks in the route graph, we combine the blocking times explained in Section 2.2 and the intuition from the reduction to @SIPP from Hanou et al. (2024). In the NS'54 signaling system, a train reserves the blocks that it is about to enter. For an upcoming block, the time at which the reservation is made is the approach time. The occupation time for a block is the time the train has fully left that block. The start of the unsafe interval per block is then equal to the moment a train enters the block k blocks in front, minus the setup and reaction time, and stops when the train has left the block plus the release time. The number k is set for how many blocks in front are reserved for a train, set by ProRail. This is dependent on the length of the block and the braking distance. For a block, the end of the unsafe interval is the end of the occupation time plus the release time of the signal (Goverde et al., 2013). For trains traveling in the opposing direction, this unsafe interval is exactly equal. This is described in Equation 4.1. The parameter t_{setup} is equal to the signal setup time, $t_{sight+reaction}$ is the sight and reaction time of the driver, and $t_{release}$ is equal to the release time of the signal. $\ell(x)$ and $\lambda(a)$ are the lengths of the block and the train, respectively. With $\nu_{avg}(x, a)$ we denote the average velocity of a train in a block, accounting for acceleration and deceleration. Lastly, t_s^x and t_e^x are the beginning and end times of when an agent is inside a block x, respectively.

$$i_x^a = \left\langle t_s^{x-k} - t_{setup} - t_{sight+reaction},\ t_e^x + t_{release} \right\rangle \tag{4.1}$$

To compute this unsafe interval for a block, we can split the unsafe interval into $k + 1$ parts, where one part is the occupation time of one block. This comes from the intuition that the approach time of a block includes the occupation time for the upcoming k blocks. The calculation for only the occupation time is shown in 4.2. These k unsafe intervals can then be merged, making sure to account for t_{pre} and t_{post} in the final unsafe interval.

$$\hat{i}_x^a = \left\langle t_s^x, t_s^x + \frac{\ell(x) + \lambda(a)}{\nu_{avg}(x, a)} \right\rangle \tag{4.2}$$

We calculate the average velocity of a train accounting for the acceleration time $acc(a)$. It is important to account for the acceleration as trains can have long acceleration times. When accelerating during a block, there are

two possibilities: the train accelerates during the whole duration of the block, or the train reaches the maximum allowed velocity $\nu_{max}(x,a)$ and stops accelerating. To know if the train reaches its maximum velocity, we compare the length of the block $\ell(x)$ with the length of a block that would be needed to reach $\nu_{max}(x,a)$. Equation 4.3 shows how to calculate this distance given a constant acceleration.

$$\ell_{min} = \frac{\nu_{max}(x,a)^2 - \nu_0(x,a)^2}{2 \cdot acc(a)} \tag{4.3}$$

If ℓ_{min} is larger than $\ell(x)$, the train accelerates during the whole block. We calculate the average velocity by averaging the starting velocity and the velocity the train reaches at the end of the block. When we do reach $\nu_{max}(x,a)$ during the block, the length of the block over the time to accelerate plus the time at maximum speed is used to calculate the average velocity.

$$\nu_{avg}(x,a) = \begin{cases} \dfrac{\nu_0(x,a) + \sqrt{\nu_0(x,a)^2 + 2 \cdot acc(a) \cdot \ell(x)}}{2} & \texttt{if } \ell_{min} \geq \ell(x) \\ \dfrac{\ell(x)}{\dfrac{\nu_{max}(x,a) - \nu_0(x,a)}{acc(a)} + \dfrac{\ell(x) - \ell_{min}}{\nu_{max}(x,a)}} & \texttt{else} \end{cases} \tag{4.4}$$

There is always an overlap of $\frac{\lambda(a)}{\nu_{avg}(x,a)}$ between two consecutive blocks in occupation time. To show this, take a train traveling between two consecutive blocks x and $x+1$. The occupation time of x is equal to

$$\langle t_s^x, t_e^x \rangle = \left\langle t_s^x, t_s^x + \frac{\ell(x) + \lambda(a)}{\nu_{avg}(x,a)} \right\rangle. \tag{4.5}$$

The occupation time in location $x+1$ starts the moment the first part of the train enters the block, that is, the time it takes to travel from the beginning of x to $x+1$. This is simply the length of the block, $\ell(x)$, over the train speed $\nu_{avg}(x,a)$. It follows that the occupation time of $x+1$ is

$$\left\langle t_s^{x+1}, t_e^{x+1} \right\rangle = \left\langle t_s^x + \frac{\ell(x)}{\nu_{avg}(x,a)}, t_s^x + \frac{\ell(x) + \ell(x+1) + \lambda(a)}{\nu_{avg}(x+1,a)} \right\rangle. \tag{4.6}$$

The overlap between the two occupation times is

$$t_e^x - t_s^{x+1} = t_s^x + \frac{\ell(x) + \lambda(a)}{\nu_{avg}(x,a)} - \left(t_s^x + \frac{\ell(x)}{\nu_{avg}(x,a)} \right). \tag{4.7}$$

It follows that the overlap between the occupation times of two consecutive blocks is equal to $\frac{\lambda(a)}{\nu_{avg}(x,a)}$.

Now let us define the operator "|" that combines two overlapping intervals and returns the interval encompassing the entire duration. Equation 4.8 shows how, given two overlapping unsafe intervals i_x^a and i_y^a, we calculate the total interval using the "|" operator.

$$\langle t_s^x, t_e^x \rangle | \langle t_s^y, t_e^y \rangle = \langle min(t_s^x, t_s^y), max(t_e^x, t_e^y) \rangle \tag{4.8}$$

The approach time $\bar{i}_x^a$ for a block x can now be calculated by combining the occupation times of the previous k blocks. This can be formally defined as $\bar{i}_x^a = \hat{i}_{x-k}^a | \hat{i}_{x-k+1}^a | ... | \hat{i}_{x-1}^a = \left\langle t_s^{x-k}, t_e^{x-1} \right\rangle$.

We can now define the blocking time for a train using the approach time and occupation time. The blocking time starts earlier than the approach time, as defined by the setup time, and the release time determines the amount of time after the occupation time that the block remains unsafe. Equation 4.9 shows how to calculate the blocking time, which is exactly equal to the length of the unsafe interval at a block, using the approach time $\bar{i}_x^a = \left\langle t_s^{x-k}, t_e^{x-1} \right\rangle$ and occupation time $\hat{i}_x^a = \langle t_s^x, t_e^x \rangle$.

$$\begin{aligned} i_x^a &= \left\langle t_s^{x-k} - t_{setup} - t_{sight+reaction}, t_e^{x-1} \right\rangle | \left\langle t_s^x, t_e^x + t_{release} \right\rangle \\ &= \left\langle t_s^{x-k} - t_{setup} - t_{sight+reaction}, t_e^x + t_{release} \right\rangle \end{aligned} \tag{4.9}$$

With Equation 4.9, we have shown that it is possible to calculate the unsafe interval for a block in parts by splitting it up into the approach time and the occupation time. In the creation of the route graph, we saw that a route can consist of multiple nodes of the track graph, and a node in the track graph can be a part of multiple routes. To correctly calculate the unsafe intervals of the routes, we calculate the occupation time for every track part in the track graph. For an occupation time $\hat{i}_u^a$ of a track part u, and for a route r of a train, if $u \in r.trackparts$ then the route r will also be unsafe for $\hat{i}_u^a$. This allows us, given a train that has a route through the track graph, to calculate the unsafe intervals for the routes of the route graph. This method also takes into account sectional releases of routes, where a route is safe after a train has moved past the switch if that route uses the other side of the switch.

As an example, we use a train leaving a platform at Hoofddorp on track W_1, going to Schiphol. It uses the route from W_1 to W_2. The track is shown in Figure 4.1a. It starts by reserving the upcoming block sections right before departure, all track parts that are parts of the upcoming k blocks will have their routes receive an unsafe interval. The route $W_1 \rightarrow W_2$ passes through the nodes $3L, 5, 8R, 11$ and W_2. The running time when a train is traveling between these nodes result in a blocking time for the route $W_1 \rightarrow W_2$ as calculated in Equation 4.9. For routes that partly intersect with the route $W_1 \rightarrow W_2$, like $V_1 \rightarrow X_2$, are also blocked for the duration

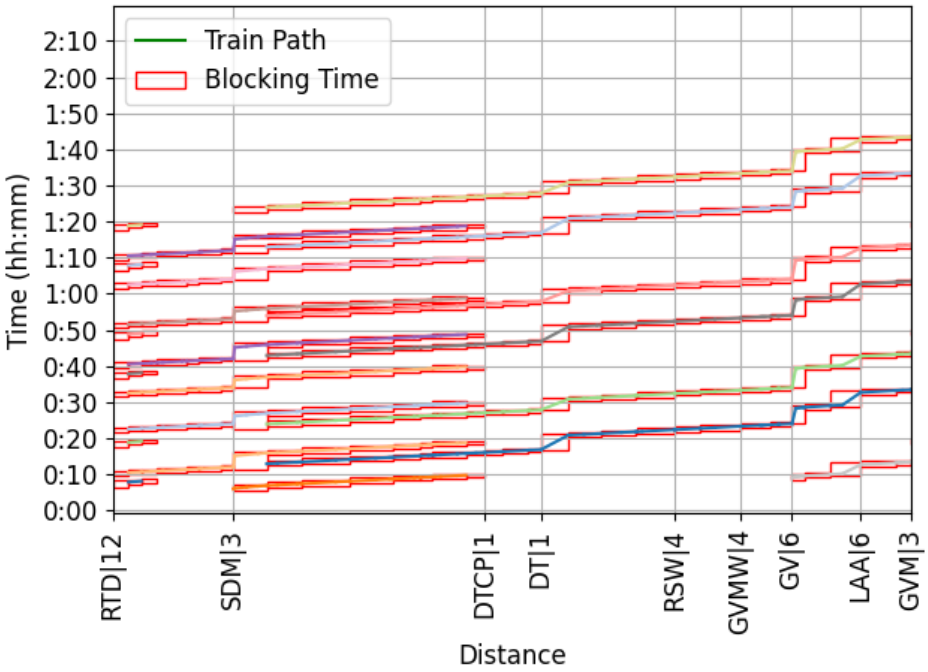


Figure 4.2: Blocking staircase diagram created in the routing graph on a single route from Rotterdam to The Hague Mariahoeve. X X-axis is showing stations with platform numbers for stations on the route.

that the intersecting node in the track graph is blocked for. In this case that is node 5. To conclude, the exact route the train has taken will have the unsafe interval as in 4.1, where other routes can also get a part of this unsafe interval, depending on how much overlap there is with the taken nodes in the track graph. By applying this blocking time calculation for a set of train on the route between Rotterdam and The Hague Mariahoeve, we get a blocking staircase diagram as seen in Figure 4.2. The track between Schiedam and Delft Campus is only one track for each direction, thus other trains driving on a track parallel merge for a bit.

4.3 Flexibility

By introducing flexibility in the arrival time function we defined in Section 2.1, we can delay agents and will be certain that no conflict is created by this action. We look at two types of flexibility. The buffer time and the recovery time. First, we will formally define the buffer time, to later supplement it with recovery time.

In this thesis, we define buffer time as the amount of time a train can be

delayed before a conflict occurs, including the time it can recover from its delay. We can calculate this using the blocking staircase diagram. Consider two trains a and b that share the same block x. That block will have two unsafe intervals $i_x^a = \langle t_s^{x,a}, t_e^{x,a} \rangle$ and $i_x^b = \langle t_s^{x,b}, t_e^{x,b} \rangle$, where b is following a on the track. At this block, we would be able to delay train a by $t_s^{x,b} - t_e^{x,a}$ until a conflict occurs, known as the local buffer time. If we were to delay an agent by this amount, there would be no conflict between the two trains. If no train occupies the same block as train a after a occupies this block, then we can delay the train indefinitely, and thus the local buffer time is infinite. The formula for $local\_bt_x^a$ is shown in Equation 4.10.

When taking the whole route of an agent into account, the buffer time can only be as large as the buffer time on the next block of the route, plus the amount of time an agent can recuperate from its delay. Thus, the amount of buffer time is limited by the local buffer time and the buffer time of the next block, plus the recovery time. We can then define the buffer time as in Equation 4.11. With this equation, we can calculate the buffer time for an agent a at a location x. The route of an agent is defined as the sequence of blocks that the agent enters sequentially. $next(x)$ returns the block in the route after x.

By using a backtracking approach, we can calculate the buffer time for all agents in $O(n * a)$ time, where n is the number of blocks on a route, and a is the number of agents. The algorithm starts at the end of a train's movements and works backwards to continuously decrease the amount of buffer time available for those blocks on the route, storing the amount of buffer time available for the current combination of block and agent.

$$local\_bt_x^a = \begin{cases} t_s^{x,b} - t_e^{x,a} & \text{if there is a train } b \text{ that follows } a \\ \infty & \text{else} \end{cases} \tag{4.10}$$

$$bt_x^a = \begin{cases} local\_bt_x^a & \text{if end of route} \\ min(local\_bt_x^a, bt_{next(x)}^a + rt_{next(x)}^a) & \text{else} \end{cases} \tag{4.11}$$

The variable $rt_{next(x)}^a$ in Equation 4.11 is the amount of recovery time the agent has in the next block. By introducing this recovery time, we are able to delay agents by a larger amount, as they can decrease their delay and be within the limits of the maximum buffer time at the next block. The recovery time is the amount of time that can be gained for an agent by traveling faster than the planned speed. This is typically 7-9% faster than the initially planned speed (Hansen and Pachl, 2008). Another form of recovery time is the duration that the train is stationary at a station. The stop can be shortened to decrease the delay. In this thesis, we define two types of recovery time. First is the recovery time at a single block for an agent, rt_x^a, which is how much time the agent can speed up in that block as

found in Equation 4.12. Second is the compound recovery time crt_x^a, which is the amount of time the agent can make up on its entire route after block x as found in Equation 4.13.

$$rt_x^a = \begin{cases} dwell\_time_x^a - minimum\_dwell\_time^a & \text{if } a \text{ stops at } x \\ \frac{\ell(x)}{\nu_{avg}(x,a)} - \frac{\ell(x)}{\nu_{avg}(x,a)\cdot 1.08} & \text{else} \end{cases} \tag{4.12}$$

$$crt_x^a = \begin{cases} 0 & \text{if end of route} \\ rt_{next(x)}^a + crt_{next(x)}^a & \text{else} \end{cases} \tag{4.13}$$

In this thesis, we assume that the impact that using recovery time has on the blocking staircase diagram, especially increasing the velocity by 7-9%, is negligible. As the velocity increases, the running time of a train in a block decreases. But as the minimum braking distance also increases, the total blocking time could also increase. The blocking time increases if k needs to increase as the breaking distance becomes larger than the block length. In other situations, the blocking time decreases.

We can apply the same backtracking approach as when calculating the buffer time to efficiently calculate the recovery time. With this approach, the calculation of the recovery time can be performed in linear time in the number of agents and is also linear in the number of moves these agents have.

Applying this algorithm to the blocking staircase diagram in Figure 4.2 results in the blocking staircase diagram found in Figure 4.3. Notice that when an agent makes a stop, the delay can be decreased by a large amount, and the flexibility increases.

4.4 Searching with flexibility

In this section, we discuss the addition to the @SIPP search algorithm by Thomas et al. (2023) to incorporate the flexibility of the moving obstacles. The introduction of flexibility increases the search space originally found in @SIPP. Initially, each location in the routing graph adds a safe interval to the problem, and each agent splits the safe intervals it passes through (Hanou et al., 2024). Before, each existing agent would add a single interval to each state and/or edge it traverses. The introduction of flexibility adds one more interval per route that the agent passes. While this does increase the search space, the problem size still scales linearly in the number of locations and agents. In Section 4.4.1 we introduce the additional edge added to the search graph. Later, in Section 4.4.2, we show the modification to the search algorithm, where we track the amount of flexibility that is used by using these additional edges.

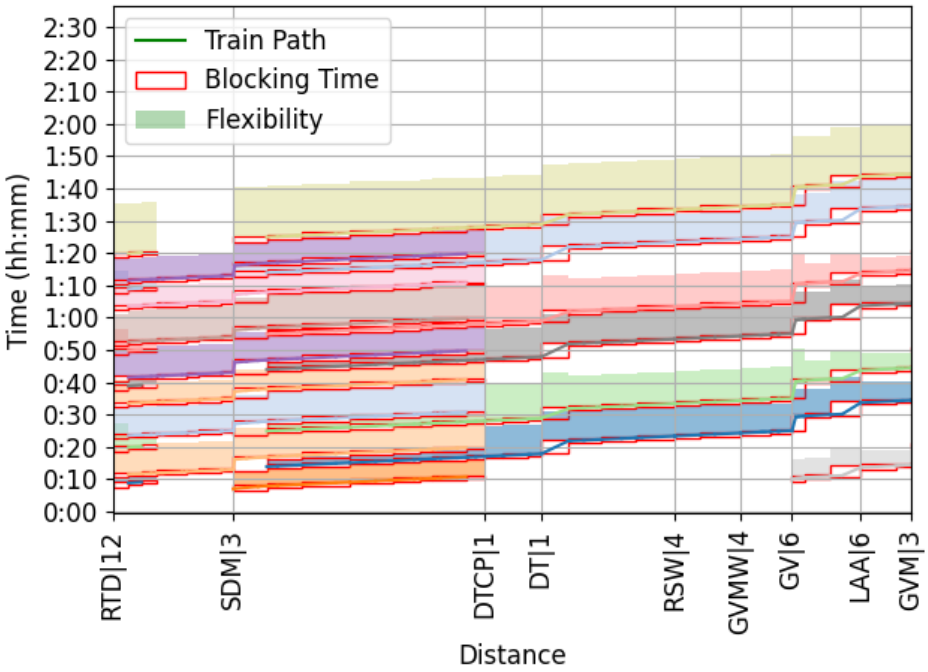
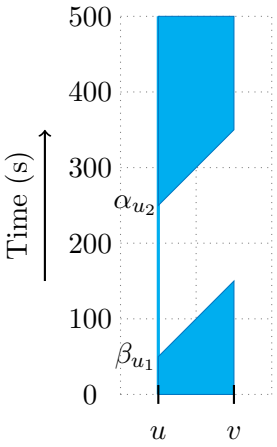


Figure 4.3: Buffer time annotated in a blocking staircase diagram of Figure 4.2. The shaded area represents the amount of delay an agent can have entering that block without impacting other agents.

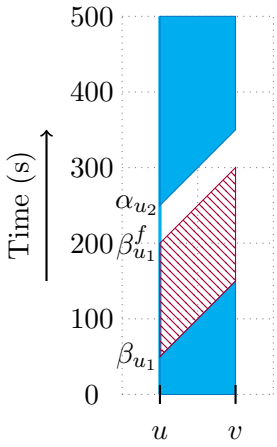
4.4.1 Additional edge

The usage of the flexibility of other agents has an impact on the arrival time function. To explain this, we look at the safe intervals of a route between u and v. In this example, there is another agent that passes through v via some other route, making it unsafe for the interval $i_v^b = \langle 150, 350 \rangle$. Let us call this agent b. This agent b splits the safe interval of the state v into two. The first interval before i_v^b, and the second safe interval after i_v^b. In @SIPP, a state is represented by a location and a safe interval, as explained in Section 2.1. The graph that this represents, we call the search graph. In this search graph, v has two nodes: v_1 with interval $\langle \leftarrow, 150 \rangle$ and v_2 with interval $\langle 350, \rightarrow \rangle$. As agent b does not enter u, it is safe to wait in u for the entire scenario. An example of this situation is shown in Figure 4.4a. Let us say that agent b has 150 seconds of flexibility in this scenario. By using this flexibility, it is now possible for agent a to arrive in v before agent b between timestep 150 and 300. This is shown by the extra striped area in Figure 4.4b.

This method of creating the safe intervals is repeated for the routing graph using the blocking staircase diagram that is calculated in Section 4.2. Every



(a) One arrival time function departs before $50s$, thus β of that ATF is $50s$, and one departing after $250s$.



(b) The striped area shows the impact of flexibility, adding more possible departure times between $50s$ and $200s$ by delaying the agent that would normally occupy that space.

Figure 4.4: Visualization of two arrival time functions for agent a. The blue area indicates when a movement from u to v is possible.

edge in the routing graph also has an associated safe interval, calculated using the same method.

Using these safe intervals, the arrival time functions can be calculated. To include flexibility in these ATFs, we introduce an extra parameter, γ^b. This denotes the maximum amount of flexibility that can be used of agent b by the ATF. The calculated arrival time functions are the edges in the search graph, between the split nodes of the routing graph. The nodes of the search graph encode if the found route is before or after agent b, as going to node v_1 is before, and node v_2 is after the blocking time of agent b. Using the flexibility of agent b is a new edge in this graph, as shown in Figure 4.5. By using the flexibility, we arrive before b in v, thus the edge goes to v_1.

4.4.2 Limits on flexibility

If the edge with flexibility is used to go to v_1, it uses between 0 and 150 seconds of flexibility of agent b, depending on the eventual actual departure time. How much time the agent b can recover from this delay later in its route is specified by the recovery time. When we explore this extra arrival time function further, we cannot decrease the delay of agent b more than the difference in compound recovery time between the current block and the block where the delay was introduced.

To keep track of the amount of flexibility that is used of other agents

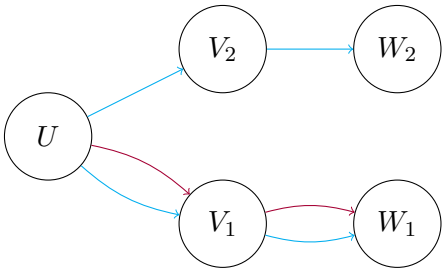


Figure 4.5: Search graph of FlexSIPP, the intervals are shown in Figure 4.6. Burgundy edges denote the extra added edges by FlexSIPP. V and W have two safe intervals, this is represented by two nodes.

during the search, we introduce two variables $\gamma^b_{min,p}$ and $\gamma^b_{max,p}$ for every existing agent b that has some flexibility. These variables are included in the definition of the arrival time function of a path p, as explained in Section 2.1. For a single agent b, the variable $\gamma^b_{min,p}$ tracks the amount of flexibility used by agent a if agent a departs at time α at a node in the search graph. This is thus the minimum amount of flexibility that needs to be used to make this path feasible. Equation 4.14 shows how to iteratively update $\gamma^b_{min,p}$ for one agent b during the heuristic search. With the term crt^b_Δ, we denote the amount of time an agent b can recover between the last edge that agent has used some flexibility and the current edge. The parameter $\gamma^b_{min,p}$ is only able to decrease by crt^b_Δ if the agent is using more flexibility than required at the current location.

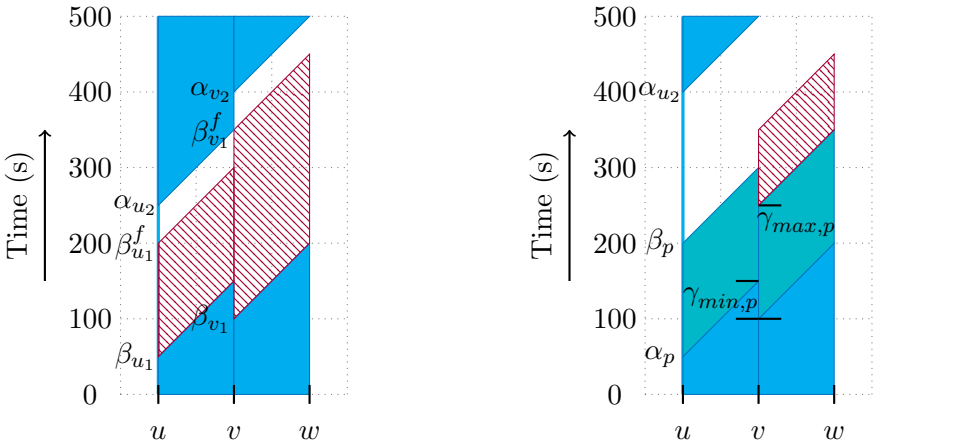
$$\gamma^b_{min,p'} = max(\gamma^b_{min,p} - crt^b_\Delta, \alpha_p + \Delta_p - \beta_e) \tag{4.14}$$

With the variable $\gamma^b_{max,p}$, we track the amount of flexibility needed when departing at the latest possible time from the origin node, β_p. The interval for how long this path is available, $max(\beta_p - \alpha_p, 0)$, defines the maximum amount of flexibility used. We also need to account for the minimum amount of flexibility used, thus $\gamma^b_{max,p} = max(\beta_p - \alpha_p, 0) + \gamma^b_{min,p}$. If α_p is bigger than β_p, it means that along the route the agent has to wait, and for any departure time of the path, it will arrive at the same time. In this case, $\gamma^b_{max,p}$ will be equal to $\gamma^b_{min,p}$.

Initially, the variables $\gamma^b_{min,p}$ and $\gamma^b_{max,p}$ are equal to 0 for every $b \in \mathcal{A}$. When traversing an edge that uses extra flexibility, $\gamma^b_{min,p}$ is set equal to the current $\gamma^b_{max,p}$, and $\gamma^b_{max,p}$ is equal to bt^a_x, the maximum amount of flexibility available at the current edge. For subsequent nodes, we can use Equation 4.14 to calculate $\gamma^b_{min,p}$. Any time the β_p is decreased, we reduce $\gamma^b_{max,p}$ for every agent that has used some flexibility by the amount that we reduce β_p.

It is no longer possible for an agent to arrive at those nodes at a time when the original amount of $\gamma^b_{max,p}$ was required. Equation 4.15 shows how to update this flexibility recursively per agent.

$$\gamma^b_{max,p'} = \gamma^b_{max,p} - max(\beta_p - \beta_{p'}, 0)) \tag{4.15}$$



(a) Initial scenario that does not use any flexibility yet. The planning agent is slower than the initial agent. Between u and v there is $150s$ of flexibility, and increases to $250s$ between v and w.

(b) When taking the edge with 0 to 150 seconds of flexibility, it extends how long the edge between v and w is available by 150 seconds, shown in turquoise.

Figure 4.6: Example of how the usage of flexibility impacts later arrival time functions

Figure 4.6 shows a situation where $\gamma^b_{min,p}$ increases. In the current state, agent a decides to take the added edge from u to v using between 0 and 150 seconds of flexibility of agent b. The earliest possible arrival time at v would be the earliest departure time at u plus the duration of the edge between u and v. This results in an arrival in v at timestep $50 + 100 = 150$. To now make the transition to w, we have two options. The turquoise area shows the edge that does not use any additional flexibility. Without any usage of flexibility, the safe period to depart would have ended at timestep 100. Since agent b is now delayed by at most $\gamma^b_{max,p}$, the ATF originally ending at timestep 100, is extended for 150 seconds. Using Equation 4.14, we find that we need to use a minimum of 50 seconds of flexibility to be able to depart and traverse the edge to w. Another option is to use more flexibility, departing from v after timestep 250. This would require at least 150 seconds of flexibility from b. As agent b is delayed by at most $\gamma^b_{max,p}$ in Figure 4.6b, if agent a were to use a route at a later point where agent b is

before agent a, the earliest departure time α of the ATF of that route will be delayed by $\gamma^b_{max,p}$.

Using the additional edge and keeping track of the amount of flexibility used during the heuristic search, we allow for moving obstacles to be delayed by an amount such that it does not impact other agents. This, in turn, allows for order changes of agents in the resulting plan. This brings the single-agent replanning approach closer to a multi-agent pathfinding approach.

Chapter 5

Evaluation

In this chapter, we evaluate the performance of FlexSIPP. By performing a case study on the Dutch railway network, we show that FlexSIPP can improve on the results of @SIPP by introducing flexibility in the agents. We first compare the output of FlexSIPP with the existing train handling document (TAD) shown in Figure 2.1. This allows us to show that FlexSIPP can identify tipping points when trains need to swap position. Then we evaluate the time complexity of FlexSIPP by evaluating the size of the problem and the number of agents in the network. Lastly, we evaluate the output of FlexSIPP by routing a Eurostar train between Schiphol and Rotterdam when the high-speed line is disrupted. This allows us to evaluate the performance of advising traffic controllers with conflict resolution. First, we discuss the dataset, consisting of the routing graph and the existing agents, and we discuss the experimental setup.

5.1 Data preparation

To convert the Dutch railway network into a routing graph as explained in Section 4.1, we have gained access to the data from Infra Atlas. The data contains the location of every piece of infrastructure of and next to the railway. Every section of track is defined by the two switches at either end, or a single switch and a bumper if it is the end of a track. Using the locations of the signals, we can define the routing graph. The resulting graph of the entire Dutch railway network consists of 9700 nodes with 247600 routes, giving us an average of 24.5 outgoing routes per node. This is, however, heavily skewed by shunting yards and unused parts of the railway network. As there are few signals inside these yards, and many different tracks, the number of possible routes to take to eventually reach the next signal grows exponentially. For example, at the shunting yard of Amersfoort, there are 16462 ways to traverse the yard starting from a single signal to the next. As we are interested in advising the traffic controllers of ProRail in cases

of delays or disruptions, in this thesis, we exclude these shunting yards and focus on the main carriageway. This results in an average of 2.48 outgoing routes per node, greatly reducing the search space. Figure 5.1 shows a histogram of the number of occurrences of the outgoing routes per node for only the main carriageway. It can be found that the vast majority of the nodes only have a single route, as would be reasonable to suspect, given that most signals are on a straight part of the track without any switches in between. At most, there are 25 options to go from one signal to the next on the main carriageway at Amsterdam Centraal.

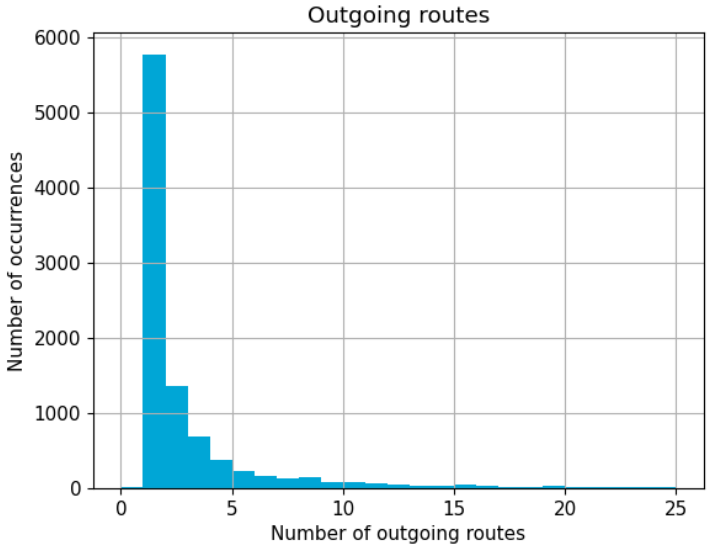


Figure 5.1: Histogram of the number of outgoing routes per node on the main carriageway on the Dutch railway network.

To calculate the blocking times of the trains in the routing graph, Table 5.1 shows the assumed parameters for the acceleration, deceleration, and minimum dwell time for the different types of trains. We further assume that all trains can reach the maximum speed of the track they are on, accounting for an 8% increase for calculating the recovery time. Lastly, we use a set length per train of 140 meters.

When traversing the diverging route of a switch, there are speed restrictions depending on the angular ratio of the switch. ProRail is aiming to standardize and reduce de number of different switches used (ProRail VenD VACO, 2016). These standard switches with accompanying speed limit can be found in Table 5.2. For non-standard switches in the railway network, we use the maximum velocity of the closest ratio of a standardized switch.

For calculating the approach time of a block, we assume that trains only need to reserve one block in front. Furthermore, we overestimate the setup time and release time combined to be at least thirty seconds. We use twenty

Train	Acceleration (m/s^2)	deceleration (m/s^2)	minimum dwell time (s)
SNG	1.1	1.1	42
SLT	0.9	1.1	42
VIRM	0.6	1.2	54
ICM	0.7	1.1	54

Table 5.1: Parameters used to calculate the running times of the trains, as well as the recovery time.

Ratio	Maximum Velocity
1:9	40 km/h
1:12	60 km/h
1:15	80 km/h
1:18	80 km/h
1:29	140 km/h

Table 5.2: Speed limit on the diverging track of a switch.

seconds for the release time of a block. We expect to overestimate the blocking times with these values, but this gives us a more robust plan if one is found.

A* search, used by FlexSIPP, can use a heuristic to guide the search into areas of the graph that are more promising. As a heuristic, we use the time to reach the destination, not considering the safe intervals. This heuristic never overestimates the cost of reaching the destination, making it admissible and ensuring that we still always find the fastest path to the destination.

Initially, we focus on the ATB using trackside signals. This is the most common system in use in the Netherlands today. Only a few corridors, like the HSL line and the Betuweroute, are currently fitted with ETCS Level 2.

The train data in the following experiments was gathered from the "Reisinformatie API". This API provides the actual and planned arrival and departure times at all stations on the route of a train. As this uses actual current timetable data, it was only gathered when there were no disruptions on the part of the network we sampled data from. This API provides the departure time at certain platforms for trains in a limited time window. The route between stops of a train was determined using the shortest path between the stations.

With this data, we have a routing graph of the Dutch railway network with an existing multi-agent plan, the railway timetable. This allows us to evaluate if we can use FlexSIPP to replan certain agents within this timetable to find tipping points for TADs, or plan new trains when disruptions occur.

The code for this project is available on GitHub (Kemmeren, 2025). This included the data of the train schedule gathered from the NS API. The code also includes a method to convert the data from Infra Atlas to the same format used by Hanou et al. (2024). The data from Infra Atlas will not be made available, but a small single-track toy problem is provided.

5.2 TAD Comparison

To determine the accuracy of the results of FlexSIPP, we compare the results of FlexSIPP with @SIPP and the train handling documents at three locations. The comparison with @SIPP shows us the effect if we do not wish to delay other trains. The locations are based on their complexity and differentiating features. The first location is the track between Schagen and Den Helder in North Holland. The railway network between these stations consists of a single track with two stations in between. At these stations, there is an additional track such that the trains traveling in opposite directions can cross. This is a relatively small problem for FlexSIPP to compare to the train handling document. The second location is between The Hague and Schiphol. This is a location with more trains and busier stations, which include sections of the railway that are four tracks wide and sections that are two tracks wide. The last location is the track between Zwolle and Meppel, which is often described as the bottleneck to and from the north of the Netherlands. It only consists of two tracks that all trains coming from and going to the north have to pass through. In Appendix A, the railway network of some sections of these locations is shown from SporenplanOnline (2025).

5.2.1 Schagen to Den Helder

The train handling document that applies to this area dictates the point where two trains of the same series, traveling in opposite directions, cross. There are four options: before entering the one-track-wide section at Schagen, at the first station Anna Paulowna, at the second station Den Helder Zuid, or at the end of the line, Den Helder. Under circumstances where there are no delays, the two trains cross at Anna Paulowna. The train handling document specifies that if the train traveling to Den Helder is delayed more than seven minutes, the crossing takes place at Schagen. This implies that with a delay of up to six minutes, it delays the train in the opposite direction at Anna Paulowna.

To test the implementation of FlexSIPP, we will find the arrival time function of a train if it were to depart at Schagen with a delay. FlexSIPP finds the shortest route to the destination, set at Den Helder, without stopping at the stations in between. In total, the scenario is run four times, three on the 21st of July, and one on the 22nd. For simplicity, we focus

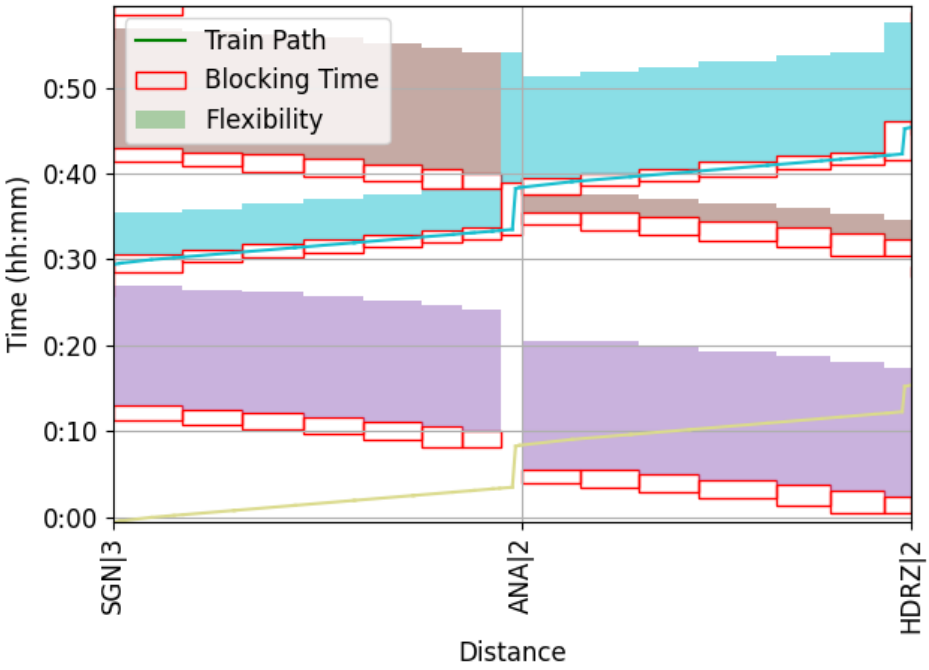


Figure 5.2: Blocking staircase diagram when replanning the first train from 3000*e*, including the flexibility that the agents have in this scenario. The original route of the train to be replanned is shown without blocking times.

on only the first scenario, but all results were within a small margin of 30 seconds the same. Figure 5.2 shows the blocking staircase diagram for this scenario, which shows that the original crossing place is at Anna Paulowna. The resulting search graph from this blocking staircase diagram for a train departing from Schagen at timestep 0 is shown in Appendix B.

In Figure 5.3, FlexSIPP has found two routes that are faster than @SIPP, shown by the earlier arrival time for the same departure time. In Table 5.3, the tipping points are extracted. To explain these tipping points, we look at the situation with a delay of around five minutes. FlexSIPP and @SIPP have both found the same route, which does not delay any other agent. If we have a delay greater than six minutes, the other train uses the track between Anna Paulowna and Schagen first. FlexSIPP determined that if we delay the 3000*o* train, it will not cause a delay propagation to other trains, and it can thus safely delay this train. That is why the 3000*e* train can go first on this part of the track, delaying the 3057 at Anna Paulowna.

The last column in Table 5.3 shows that if the replanning train has a delay of 11:37, it will delay the 3057 by 4:33. As the increase in delay is assumed to be linear, we can conclude that the train handling document is specified

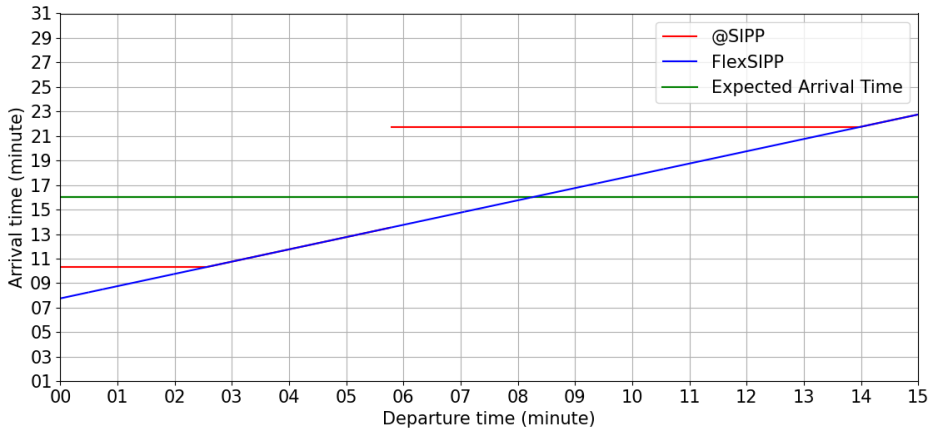


Figure 5.3: Arrival time function of a train from the series 3000*e* departing from Schagen, travelling towards Den Helder.

Train	Delay Location	Tipping Point	Delay Amount
3057	Hdrz	03:29	08:01
	Ana	11:37	04:33
	Sgn	17:18	00:00

Table 5.3: Tipping points generated by FlexSIPP for a train from the series 3000*e*, the Train column is the train it crosses with. The Delay Location denotes the location where the trains should cross if the train from the 3000*e* series is delayed for up to the amount specified in the Tipping Point column. The last column denotes the maximum amount of delay the other train will have if delayed at the delay location.

in a way to not delay the train from the series 3000*o*.

The Tipping Point column shows the absolute last possible point a route is available. FlexSIPP has determined that it can delay the opposing train by at least eight minutes. This results in an additional route where the algorithm determines that the opposing train can wait at a station earlier than in the normal timetable. The planning train can arrive a few minutes earlier by using this option, as seen in Figure 5.3. The downside is that this crossing location results in a large delay for the opposing train at the next few stations it stops at, but crucially, no oil spill effect.

5.2.2 Bottleneck between Zwolle and Meppel

Right after Zwolle, all trains to Leeuwarden and Groningen merge onto a single track per direction that lasts until after Meppel. We focus on the interaction between four train series: 600*o*, 700*o*, 6100*o*, and 9000*o*. The 600*o* and 700*o* are intercities, whereas the 6100*o* and 9000*o* are sprinters. We find the tipping points for when a train of the 600*o* series is delayed. Under standard operation, the order on the track is 600*o*, 700*o*, 6100*o*, and lastly the 9000*o*. The train handling document of this location specifies that when the delay of the 600*o* is between 5 and 10 minutes, it swaps order with the 700*o*. With a delay of 11 to 15 minutes, it also swaps with the 6100*o*. If the delay increases to more than 15 minutes, it is the last train in the order.

Figure 5.4 shows the blocking staircase diagram created for this scenario. In this blocking staircase diagram, the train we are replanning departs from Zwolle at 52 minutes. It can be seen that many trains follow each other closely in this corridor.

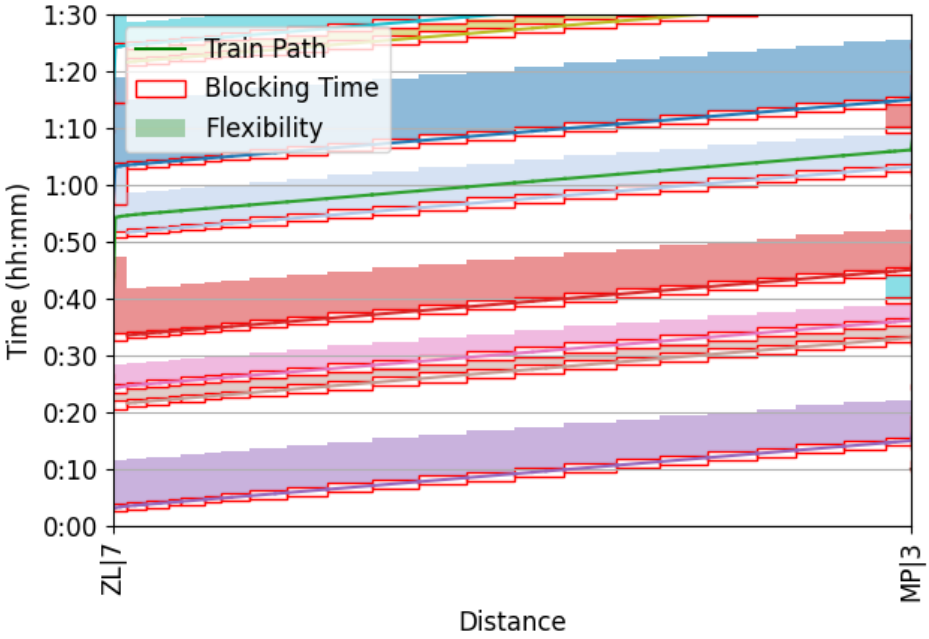


Figure 5.4: Blocking staircase diagram when replanning a train from 600*o*, including the flexibility that the agents have in this scenario. The original route of the train to be replanned is shown without blocking times.

We run FlexSIPP to find the fastest route between Amersfoort Centraal and Steenwijk, one station before and after the bottleneck. In Figure 5.5, one arrival time function is shown for this route. It shows two intervals

where it is faster to delay another train. The first interval is with a delay of four to six minutes. In this interval, it is quicker to delay a train from the series 700*o*. After this interval, FlexSIPP says the order must be swapped. The second interval is with a delay of ten to thirteen minutes. The train that is being delayed in this interval is of the series 9000*o*. FlexSIPP did not find an interval where it is better to delay an agent of the series 6100*o*.

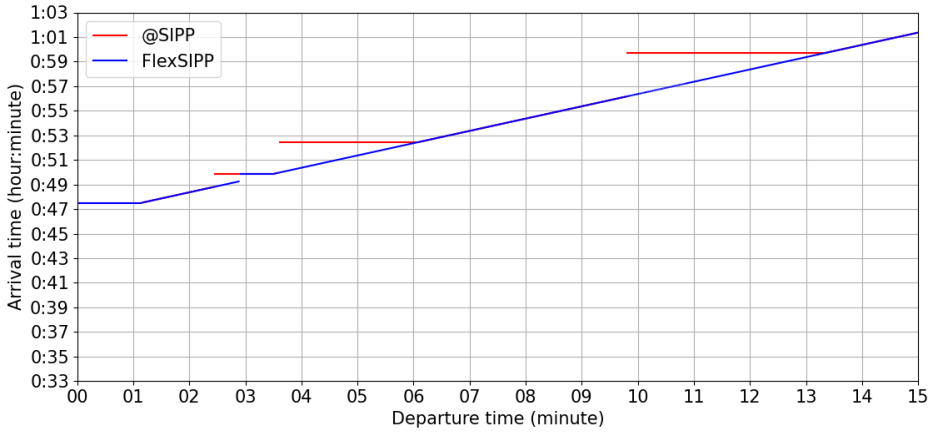


Figure 5.5: Arrival time function of a train departing from Amersfoort Centraal, traveling towards Steenwijk.

5.2.3 The Hague to Amsterdam South

The last train series we compare using a train handling document is one of the series 3500*o* between The Hague Laan van NOI and Amsterdam South. The train handling document specifies that a train of the intercity series 3500*o* swaps order with a train from the series 4300*o* if its delay is larger than seven minutes. The track between The Hague and Leiden consists of two tracks for each direction. After Leiden, it narrows to two tracks in total. This makes the order in which trains enter the track between Leiden and Hoofddorp important.

The blocking staircase diagram shown in Figure 5.6 shows the blocking time diagram between Leiden and Schiphol. Due to limitations of the NS-API, the platform that these trains use at Schiphol is incorrect.

The results of FlexSIPP are shown in Figure 5.7. It shows two intervals where a faster route is available by delaying another train. Around the six-minute mark, it delays a train from the series 4100*o*, a sprinter train starting at Leiden. This train follows the same route as a train from the series 4300*o*. The same train is delayed with a starting delay between seven and eleven minutes, but the location where the order is swapped differs. For a smaller delay, it happens later on the route at Hoofddorp. A larger delay causes the

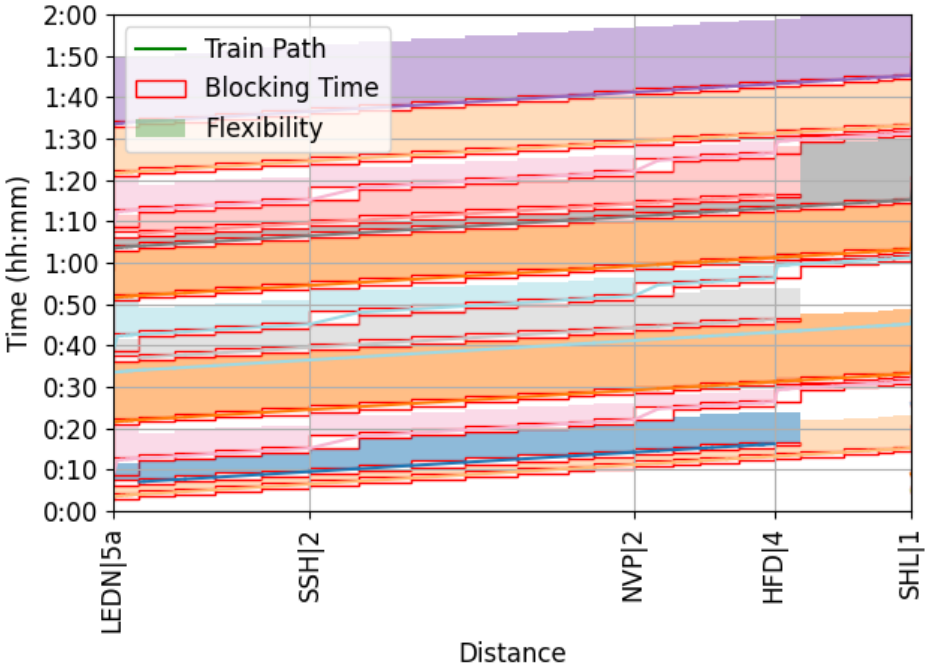


Figure 5.6: Blocking staircase diagram when replanning a train from 3500o, including the flexibility that the agents have in this scenario. The original route of the train to be replanned is shown without blocking times.

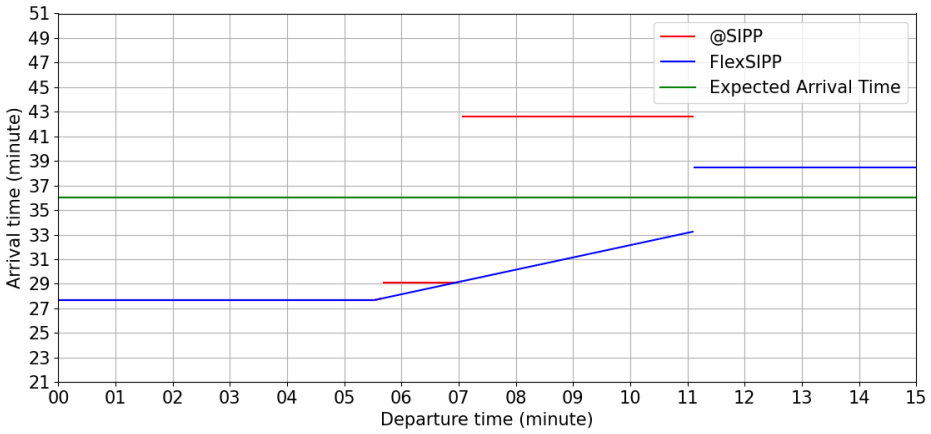


Figure 5.7: Arrival time function of a train departing from The Hague Laan van NOI to Amsterdam South.

order swap to take place at Leiden.

5.3 Runtime evaluation

It is theorized that the problem size scales linearly with the number of agents. To verify this, we perform two experiments. The first experiment determines the influence of the number of blocks of the path that the algorithm is required to plan. The second experiment identifies the connection between the number of existing agents in the network and the runtime of the algorithm. As the algorithm builds upon the implementation of A* search, we expect the worst-case time complexity of FlexSIPP to be equal to that of A* search, that is $O(b^d)$. We consider the branching factor b and the depth of the solution in the tree d.

5.3.1 Number of blocks

By increasing the number of blocks between the origin and the destination, we increase the depth of the solution in the tree. We can accomplish this by iteratively replanning a train from its origin to every station on its route. This increases the search space for every consecutive station by increasing the depth at which the solution is found. In this experiment, we replan a single intercity train on the route from Rotterdam Blaak to Amsterdam South of the series 3500*o*. It will have to route around 118 other agents that are simulated for 1 hour after the start of the simulation.

We assume that the maximum branching factor is constant between the origin and every station on the route. This allows us to compare the trend in the running time with the expected worst-case time complexity $O(b^d)$, where b is a constant and d scales with the number of blocks.

With the current implementation of @SIPP and FlexSIPP, the runtime increases drastically when multiple paths are found. To find the trend in the number of blocks, we account for this by dividing by the number of paths found.

Figure 5.8 shows the average runtime for ten scenarios. We find that with the number of points used, we can not conclusively say what the tightest bound is on the time complexity. We do find that FlexSIPP can find a valid route for a train in under 60 seconds. The outlier seen in Figure 5.8 is when planning a route to Leiden Centraal. In this instance, the algorithm had trouble finding a valid path to the specific station platform it was routing to, which led to exploring ten times the number of nodes as opposed to the previous station. For stations that are located further, FlexSIPP found a route that passes through Leiden at a different platform.

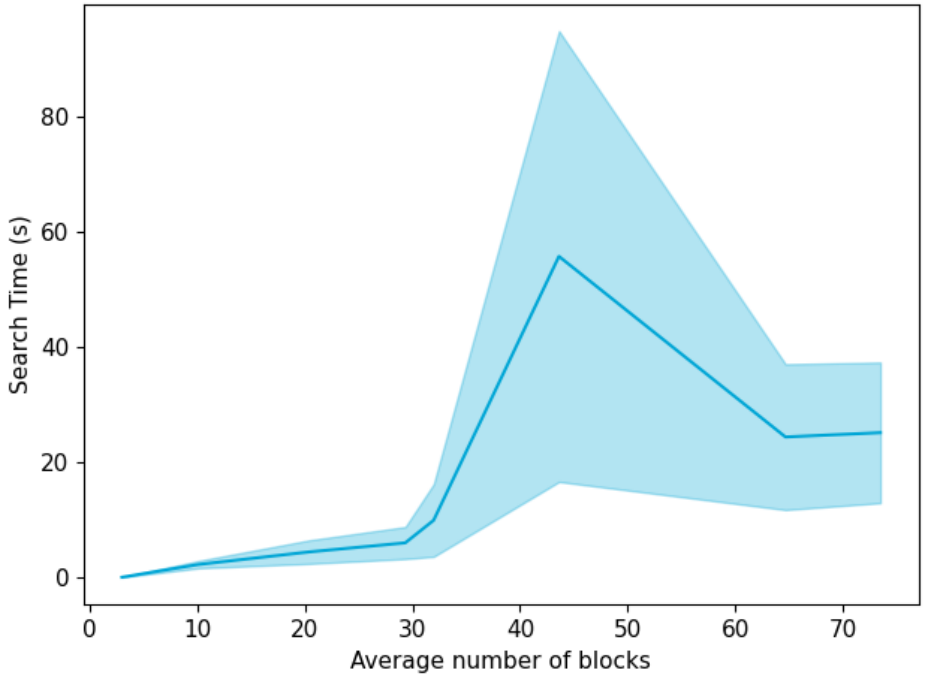


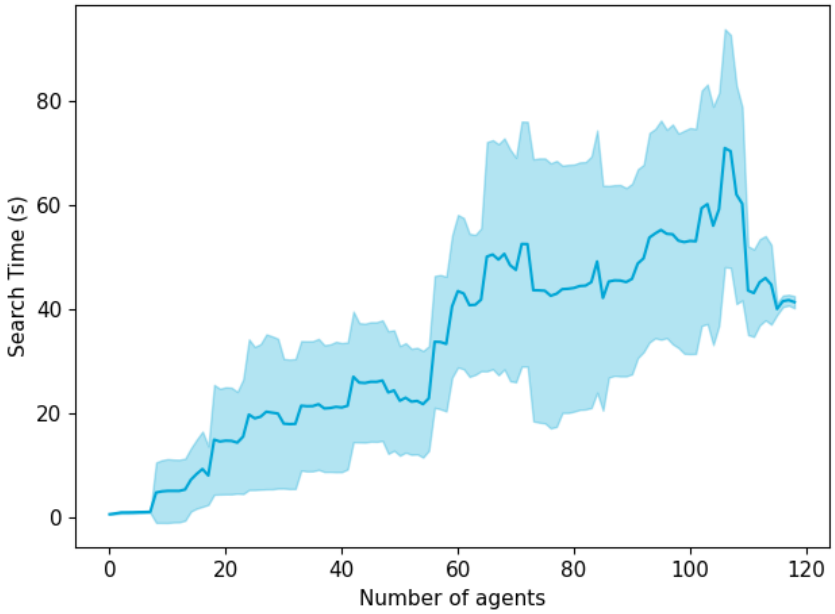
Figure 5.8: The search time of FlexSIPP over the number of blocks of the path to the destination.

5.3.2 Number of agents

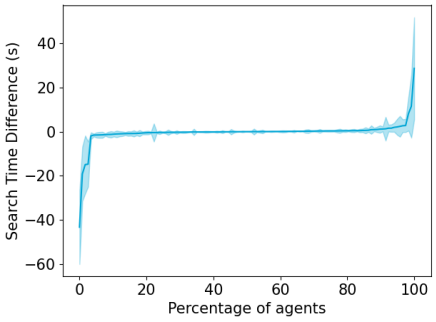
The branching factor is affected by the number of agents in the simulation, as for every agent that passes through a node in the route graph, the size of the search space increases. To measure the impact that the number of agents has, we measure the runtime of the same train of the series 3500*o* for a single state of the railway network, taken on July 22nd, 2025. We find a route from Rotterdam Blaak to The Hague HS. In total, there are 120 trains in the network. By including one train at a time, we can measure the impact that that train has on the runtime. For this experiment, we randomly create ten orderings in which the trains are included from the search.

The results in Figure 5.9a show a high variance in runtime when including more agents. To further explore the effect that certain agents have, Figure 5.9b shows the distribution of agents sorted by the average impact on the runtime.

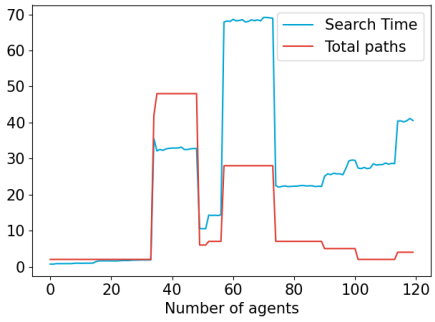
It becomes clear that most agents do not impact the runtime of the algorithm. These agents are trains that are located far away from Rotterdam, or are running at a completely different time. A few agents do have a high



(a) Runtime of FlexSIPP over the number of agents from the simulation.



(b) Distribution of the impact of including an agent from the simulation.



(c) One single run of including agents from the simulation, annotated with the total number of paths found to the destination.

Figure 5.9: Results of FlexSIPP when changing the number of agents included. Agents are added in a random order, and the experiment is repeated ten times. The shaded area represents the standard deviation.

impact on the runtime. There are four agents that, when included, increase the search time by more than ten seconds on average. Three of the four agents are sprinter trains. As these trains are relatively slow, FlexSIPP explores more of the search space to check if an order swap is possible.

Furthermore, Figure 5.9b also shows that including some agents decreases the runtime of the algorithm. Even though this is counterintuitive, it can

be explained using Figure 5.9c. It shows a single run, annotated with the total number of paths found by FlexSIPP. Because of the implementation of FlexSIPP, the runtime drastically increases with the number of paths found. The figure clearly shows that when including some agents, the number of available paths to the destination decreases, and with that also the search time.

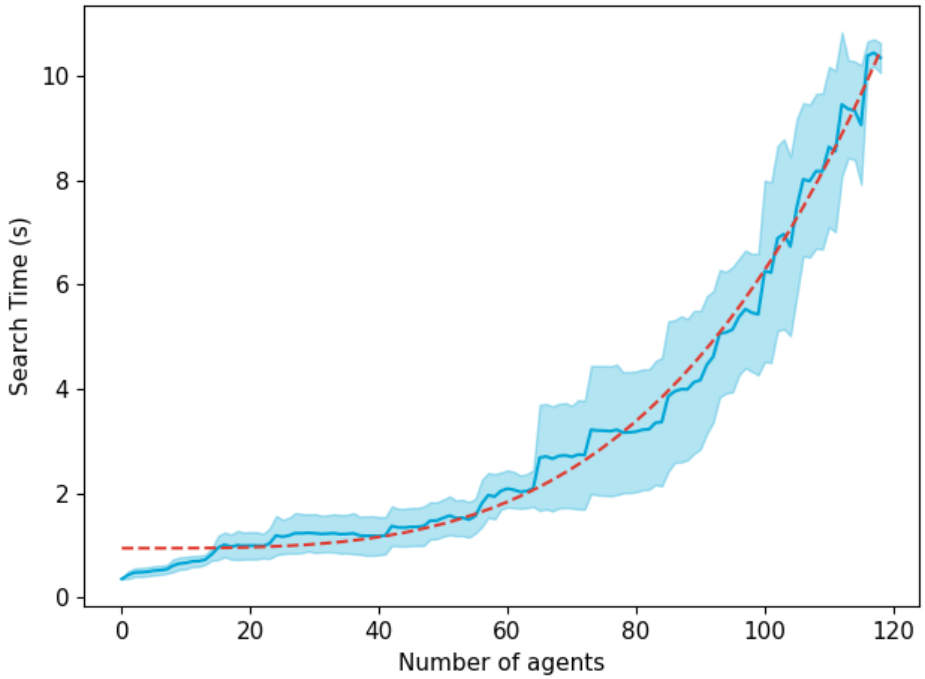


Figure 5.10: Runtime of FlexSIPP to calculate a single path over the number of agents excluded from the simulation, with a fitted polynomial function.

By calculating the runtime per path found, we get a representation of the influence of excluding agents from the search space that is closer to the theorized influence. Figure 5.10 shows that relationship. It shows a polynomial relationship between the search time and the number of agents excluded.

5.4 Routing a train from Schiphol to Rotterdam

Lastly, we look into the example we have used throughout this thesis, of finding a route for a Eurostar train between Schiphol and Rotterdam, being rerouted through The Hague and Leiden.

As this simulation includes every train in the normal timetable, it is difficult to find a valid route. It follows the same path as trains from the series

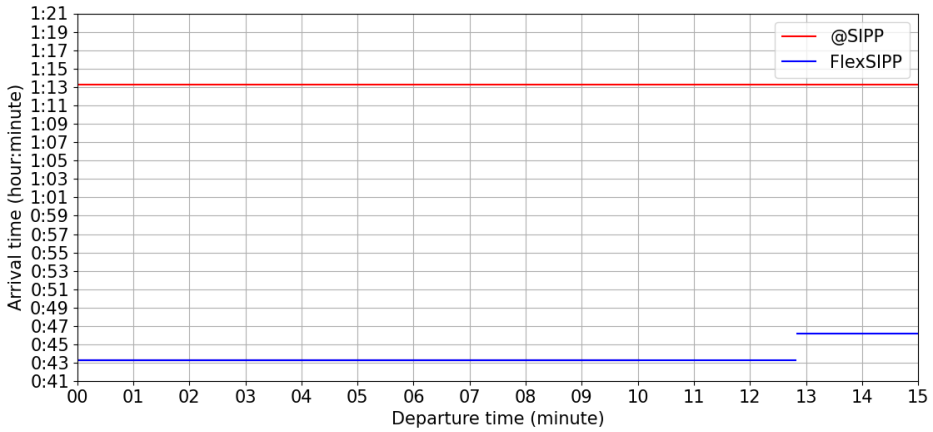


Figure 5.11: Arrival time function for planning a Eurostar train between Rotterdam and Schiphol.

3500e and 3200e, but without stopping at the stations. This causes the Eurostar train to be stuck behind Sprinter and Intercity trains, finding tipping points and locations where it can swap order is crucial in finding a feasible route.

Figure 5.11 shows the arrival time function gathered from @SIPP and FlexSIPP. FlexSIPP can find a route that is 35 minutes faster than the route gotten from @SIPP. It achieves this by swapping orders with trains at different locations. At Delft, the found route switches order with the other trains when going onto a single-track wide section per direction. At The Hague HS, the route uses a track that does not have a station platform, and the same goes for Leiden Centraal. At these locations, it delays trains located at the station platforms to find a path.

Further investigation into the route @SIPP has found shows that the route is not viable. It waits for every train in the simulation to depart to find a route that is after every other agent. In real operations, the timetable is cyclic, and the path would conflict with agents currently not simulated. This experiment has shown that we can find routes using FlexSIPP that @SIPP can not find.

Chapter 6

Discussion

In this chapter, we discuss the most important results of the experiments. Some further limitations of the study are also explained. In Section 5.2, we have shown that FlexSIPP finds routes that are comparable with train handling documents in most situations. By identifying the tipping points, we can better understand what the consequences are for choosing one path over another, as the delays for other trains become immediately clear, as well as the arrival time of the train that is planning a new route. FlexSIPP achieves this in two parts. First is describing the multi-agent execution delay replanning problem as a graph we can search on, the second is efficiently using the flexibility of agents without letting the size of the search space explode.

Firstly, we look at our multi-step reduction to a search graph. By comparing the results of FlexSIPP with the TADs, we have shown that our reduction from the railway network to the search graph follows the blocking time theory. This leads us to believe that the routing graph can be used for any signaling system that uses blocks. We have shown this reduction for NS'54, and ETCS Level 2 Virtual Block also works using blocks that are usually smaller (European Rail Supply Industry Association, 2025). For ETCS Level 2 with Hybrid Train Detection, the reduction to the search graph is also possible. The problem is that with block lengths as small as 20 meters (Versluis et al., 2025), the routing graph becomes much larger than with the average block length of 1345 meters in the current Dutch railway network on tracks that still use the NS 54 signaling system. Although the reduction of the railway network to the route graph is valid, the running time calculation can be improved by taking into account different driving strategies as performed by Wang et al. (2025). This improves the blocking time calculation, and with that comes a more accurate buffer time and recovery time calculation. More accurate running time calculations also reduce the discrepancy between the expected arrival time and the arrival time calculated by @SIPP and FlexSIPP as seen in Figures 5.3 and 5.7.

The second part of FlexSIPP that finds a route in a search graph works independently of the problem. In this thesis, we have focused on the railway sector as it is a great example of a problem where many agents need to use the same space. Comparing this to the multi-agent pathfinding literature, Švancara et al. (2019) have shown that using single-agent replanning in densely populated grids is far from optimal and can be improved upon. In this work, we take a step closer to multi-agent planning from a single-agent replanning perspective. By routing a train from Schiphol to Rotterdam in Section 5.4, we show that FlexSIPP can find more routes by introducing delays in other agents, instead of seeing them as immovable obstacles and routing around them. As the number of routes is very limited since the trains are restricted to the tracks, it is important to be able to find routes that are otherwise blocked by other agents. In other multi-agent pathfinding settings where agents have a wider range of movements, the difference between @SIPP and FlexSIPP becomes less pronounced than in Figure 5.11. Only when two agents need to occupy the same space will FlexSIPP explore the possibility for the replanning agent to enter this space first.

Furthermore, in Section 5.3 we show that FlexSIPP can efficiently search in the search graph with the added flexibility. The effect that other agents have on the search time can be unpredictable. Some agents drastically increase the search time of FlexSIPP, mostly by increasing the number of routes that are possible to the destination. As FlexSIPP finds any conflict-free route, not all of these routes are desirable. In Section 5.4, for instance, FlexSIPP has found 21 paths. Of these paths, seven travel in the opposite direction of the standard travel direction for some part of the route. This is not desired, but currently, we lack the data to automatically get the travel direction for every track. And even though it is not desired, it does not create conflicts with other trains on the track.

Lastly, we discuss a limitation of FlexSIPP. To show the time complexity of FlexSIPP in Figure 5.10, we had to account for the number of paths found. With the current implementation of FlexSIPP, the runtime drastically increases with the number of paths found. This is due to the fact that heuristic search is restarted from the beginning after a path is found. The large increase in the number of paths found in Figure 5.9c can be partly attributed to planning a slow train in front of a faster train. It first finds a path using a bit of flexibility, but only available for a bit. It then restarts using a bit more flexibility of that same train but at a later block. This is repeated many times. The reason for this restart being implemented, is pruning paths that are worse than the path found. Using a smarter and more efficient pruning strategy will help in improving the performance of FlexSIPP.

Chapter 7

Conclusion

Our aim in this thesis is to show that using the flexibility of moving obstacles in any-start-time safe interval path planning can improve the cost function in a multi-agent pathfinding problem. In this problem, the other agents are moving obstacles that we can manipulate and delay by some amount, specified by their flexibility. This flexibility is defined in such a way that if the agents are delayed by this amount, they still have a conflict-free route.

The experiments have shown that by exploiting this flexibility, we can explore more of the search space of the multi-agent problem while viewing the problem as a single-agent replanning problem. It allows for using heuristic search to efficiently find a new route for a single agent. The representation of the additional edge in the search graph that any-start-time safe interval path planning uses makes sure that the search space does not grow too fast, and finding a route using flexibility is fast.

By conducting a case study on the Dutch railway network, we have shown a method to reduce the railway network to a directed graph that can use blocking time theory to determine the occupation of the network. We have shown that this representation can be used to plan new trains by determining the safe intervals between the blocking times of other trains. This allows us to find conflict-free paths in a busy railway network like the Dutch railway network. This reduction to a graph can be used to model signalling systems that use blocks, if accurate runtime and blocking time calculations are used.

To determine the validity of the introduced flexibility in any-start-time safe interval path planning, we compared the generated tipping points with the train handling documents. FlexSIPP can determine the same action and amount of delay specified in these documents, together with the consequences for other agents in the network. As FlexSIPP is able to identify tipping points automatically and provide the effect that departing before or after this tipping point has, it can aid traffic controllers during conflict resolution. We are able to construct TADs dynamically and quickly for the current situation of the railway network to advise traffic controllers.

Chapter 8

Future Works

In this chapter, we discuss two directions this research can be taken further. First is applying the methods discussed in this thesis in relation to the railway sector. This is partly in the ongoing research into ETCS Level 2, and partly in the operability of the provided solutions, considering the human factor in the current day-to-day operations at ProRail. A second direction this research can be taken is exploring the possibilities of flexibility in other fields of research.

The current reduction from the railway network to a graph we used is based on the most common signaling system in use today. Current research in this sector is looking ahead at the system that will be implemented in the coming years: ETCS Level 2 with trackside train detection (Versluis et al., 2024). By implementing a reduction in this new system to a graph with accurate running times, FlexSIPP will be able to work with this system. A more difficult approach would be the introduction of ETCS Level 2 with hybrid train detection (Versluis et al., 2025). The difficulty is in stepping away from the blocks that the current network is made out of and implementing a moving block system. One approach would be going back to the original reduction to a graph by Hanou et al. (2024), and augmenting it such that overtaking on a long straight part of the track is not possible. Extra care also needs to be taken with calculating the headway of trains in this situation. If an accurate reduction is made, the search algorithm of FlexSIPP will work with this new system.

Improvements in advising traffic controllers can be made by accounting for the human factor. FlexSIPP currently finds any feasible route by allowing the agent to change tracks or even run in the opposite direction of travel. Even though these tracks are feasible, it is unlikely that a traffic controller will direct a train to use such a route. By investigating the preferences of traffic controllers and incorporating this, FlexSIPP will be able to create feasible routes that conform to the quality of solutions that traffic controllers prefer.

Current state-of-the-art mixed integer linear programming models model the routes of trains accurately, accounting for driver behaviour. A method for incorporating this in FlexSIPP is using information from simulators like FRISO (Middelkoop and Loeve, 2006). Using blocking times from such an application will cause the accuracy of FlexSIPP to increase, making sure that the routes found are actually viable. This would also allow for the creation of new train handling documents. It also allows for creating turnkey solutions for traffic controllers in a dynamic setting. This solves the problem that traffic controllers currently have, where multiple TADs are applied at the same time. By stacking TADs, it can happen that they are applied in a way they were not designed for. By generating new TADs dynamically based on the current situation of the railway network, this problem is solved.

The usage of the flexibility of agents is not limited to the replanning of trains. In this thesis, we have shown the application of FlexSIPP on the Dutch railway network. FlexSIPP can be applied to problems where there exists a plan that needs to be modified. This can include planning a new agent or replanning an existing agent. Warehouse robot scheduling (Wang, 2024) is very similar, where the agents are robots picking orders in a warehouse. When a new order with high priority arrives, FlexSIPP can be applied to fulfill the order as fast as possible without causing problems for other robots. In such a problem where there are many agents, but still room for flexibility in these agents, we expect that FlexSIPP will perform well in creating a plan for a new or delayed agent.

To quantize the results of FlexSIPP, we can extend the work of Švancara et al. (2019) discussed in Section 3.1. By comparing FlexSIPP against replanning all agents, it is possible to show the loss in optimality versus the run time. FlexSIPP will be able to improve the overall goal, while still being "extremely fast" as single-agent replanning.

Bibliography

Atzmon, D., Stern, R., Felner, A., Wagner, G., Bartak, R., and Zhou, N.-F. (2018). Robust Multi-Agent Path Finding. *Proceedings of the International Symposium on Combinatorial Search*, 9(1):2–9. Number: 1.

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.

Chen, Z., Harabor, D. D., Li, J., and Stuckey, P. J. (2021). Symmetry Breaking for k-Robust Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12267–12274.

European Rail Supply Industry Association (2025). ERTMS Signaling levels. https://www.ertms.net/ertms-signaling-levels/.

Fang, W., Yang, S., and Yao, X. (2015). A Survey on Problem Models and Solution Approaches to Rescheduling in Railway Networks. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2997–3016.

Fischetti, M., Salvagnin, D., and Zanette, A. (2009). Fast Approaches to Improve the Robustness of a Railway Timetable. *Transportation Science*, 43(3):321–335. Publisher: INFORMS.

Goverde, R. M. P. (2007). Railway timetable stability analysis using max-plus system theory. *Transportation Research Part B: Methodological*, 41(2):179–201.

Goverde, R. M. P., 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 & Management*, 3(3):78–94.

Hanou, I. K., 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. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34:258–266.

Hansen, I. A. and Pachl, J. (2008). *Railway timetable and traffic : analysis, modelling, simulation*. EurailPress, Hamburg. Section: 228 p.

Kemmeren, E. (2025). EricKemmeren/delay-replanning-slack. https://github.com/EricKemmeren/delay-replanning-slack.

Kersbergen, B., Rudan, J., van den Boom, T., and De Schutter, B. (2016). Towards railway traffic management using switching Max-plus-linear systems. *Discrete Event Dynamic Systems*, 26(2):183–223.

Middelkoop, A. D. and Loeve, L. (2006). Simulation of traffic management with FRISO. In *Computers in Railways X*, volume 1, pages 501–509, Prague, Czech Republic. WIT Press. ISSN: 1743-3509, 1746-4498.

Nebel, B. (2024). The computational complexity of multi-agent pathfinding on directed graphs. *Artificial Intelligence*, 328:104063.

NS (2025). Spoorkaart 2025. https://nieuws.ns.nl/spoorkaart-2025-hier-te-downloaden/.

Pachl, J. (2020). *Railway Signalling Principles*. Technical University Braunschweig, Germany.

Phillips, M. and Likhachev, M. (2011). SIPP: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635. ISSN: 1050-4729.

ProRail (2024). Network Statement 2025. https://www.prorail.nl/samenwerken/vervoerders/network-statement.

ProRail (2024). Spoorkaart. https://www.prorail.nl/reizen/spoorkaart.

ProRail VenD VACO (2016). Regels voor het functioneel ontwerp van railinfrastructuur.

Quaglietta, E., Pellegrini, P., Goverde, R. M. P., Albrecht, T., Jaekel, B., Marlière, G., Rodriguez, J., Dollevoet, T., Ambrogio, B., Carcasole, D., Giaroli, M., and Nicholson, G. (2016). The ON-TIME real-time railway traffic management framework: A proof-of-concept using a scalable standardised data communication architecture. *Transportation Research Part C: Emerging Technologies*, 63:23–50.

Rudan, J., Kersbergen, B., van den Boom, T., and Hangos, K. (2013). Performance analysis of MILP based model predictive control algorithms for dynamic railway scheduling. In *2013 European Control Conference (ECC)*, pages 4562–4567.

SporenplanOnline (2025). SporenplanOnline. https://www.sporenplan.nl/.

Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K., Barták, R., and Boyarski, E. (2019). Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Proceedings of the International Symposium on Combinatorial Search*, 10(1):151–158.

Thomas, D., Shimony, S., Ruml, W., Karpas, E., Shperberg, S., and Coles, A. (2023). Any-Start-Time Planning for SIPP. *Proceedings of the ICAPS-23 Workshop on Heuristics and Search for Domain-Independent Planning.*

Van den Boom, T. J. J., Weiss, N., Leune, W., Goverde, R. M. P., and De Schutter, B. (2011). A permutation-based algorithm to optimally reschedule trains in a railway traffic network. *IFAC Proceedings Volumes*, 44(1):9537–9542.

Versluis, N. D., Pellegrini, P., Quaglietta, E., Goverde, R. M. P., and Rodriguez, J. (2024). Conflict Detection and Resolution for Distance-to-Go Railway Signalling.

Versluis, N. D., Pellegrini, P., Quaglietta, E., Goverde, R. M. P., and Rodriguez, J. (2025). Impact of track discretisation on conflict detection and resolution under ETCS with onboard train integrity monitoring. *Journal of Rail Transport Planning & Management*, 35:100533.

Wang, A. (2024). Intelligent warehouse multi-robot scheduling system based on improved A* algorithm. In *2024 3rd Conference on Fully Actuated System Theory and Applications (FASTA)*, pages 1305–1310.

Wang, Z., Quaglietta, E., Bartholomeus, M., and Goverde, R. M. P. (2025). Sensitivity of Train Path Envelopes for Automatic Train Operation. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13.

Yu, J. and LaValle, S. M. (2013). Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, pages 1443–1449, Bellevue, Washington. AAAI Press.

Zhou, R. (2023). Adding flexibility to the timetable in real-time railway traffic management. Master's thesis, TU Delft.

Švancara, J., Vlk, M., Stern, R., Atzmon, D., and Barták, R. (2019). Online Multi-Agent Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7732–7739.

Appendix A

Sporenplan Railway Networks

This section shows some of the locations that the train handling documents are about.

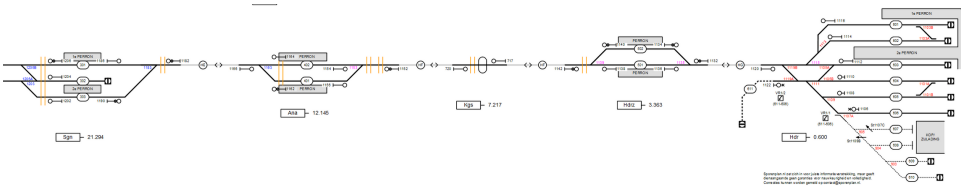


Figure A.1: Sporenplan image from Den Helder, showing two locations in the single track section where trains can cross.

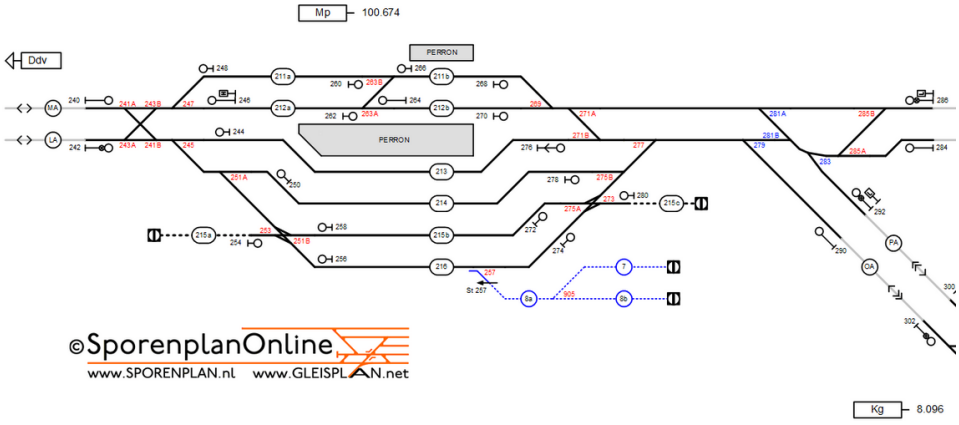


Figure A.2: Sporenplan image from Meppel. Only after Meppel do the two tracks divert. Between Zwolle and Meppel is only two tracks wide.

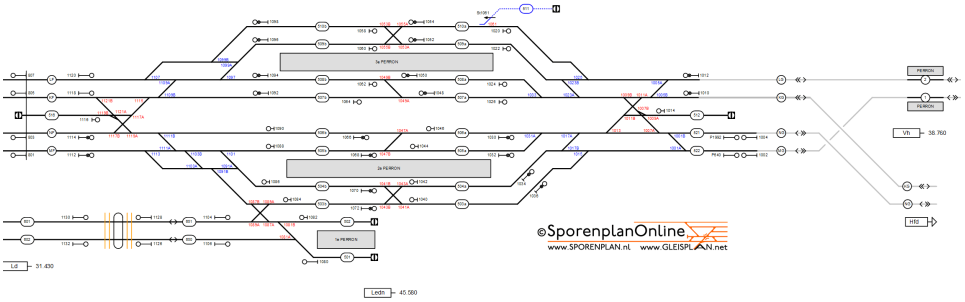


Figure A.3: Sporenplan image from Leiden. From the four incoming tracks from The Hague (top left), two go to Hoofdorp, and two go to Haarlem.

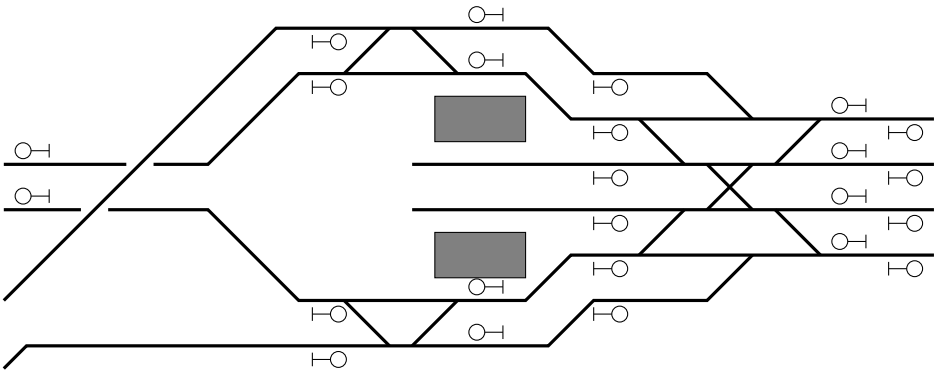


Figure A.4: Railway network at Hoofddorp, the HSL line and line from Leiden merge going to Schiphol to the right.

Appendix B

Search graph

This section shows the search graph of Den Helder that is created from the blocking time diagram of Figure 5.2. Figure B.1 shows a section of the most common routes to platform 1 of Den Helder. Some nodes are omitted: it would technically be possible to turn around at another platform, travel away from Den Helder to turn around again somewhere else, to then finally go to platform 1 of Den Helder. There are also at most 2 nodes per location, as the starting node becomes blocked after 30 minutes when the next train of the same series is at Schagen.

One remark about Figure B.1 is that the cyan edges, which denote the edges without using flexibility, are missing from s_1^9. The reasoning is that without using flexibility, an agent can't use this edge.

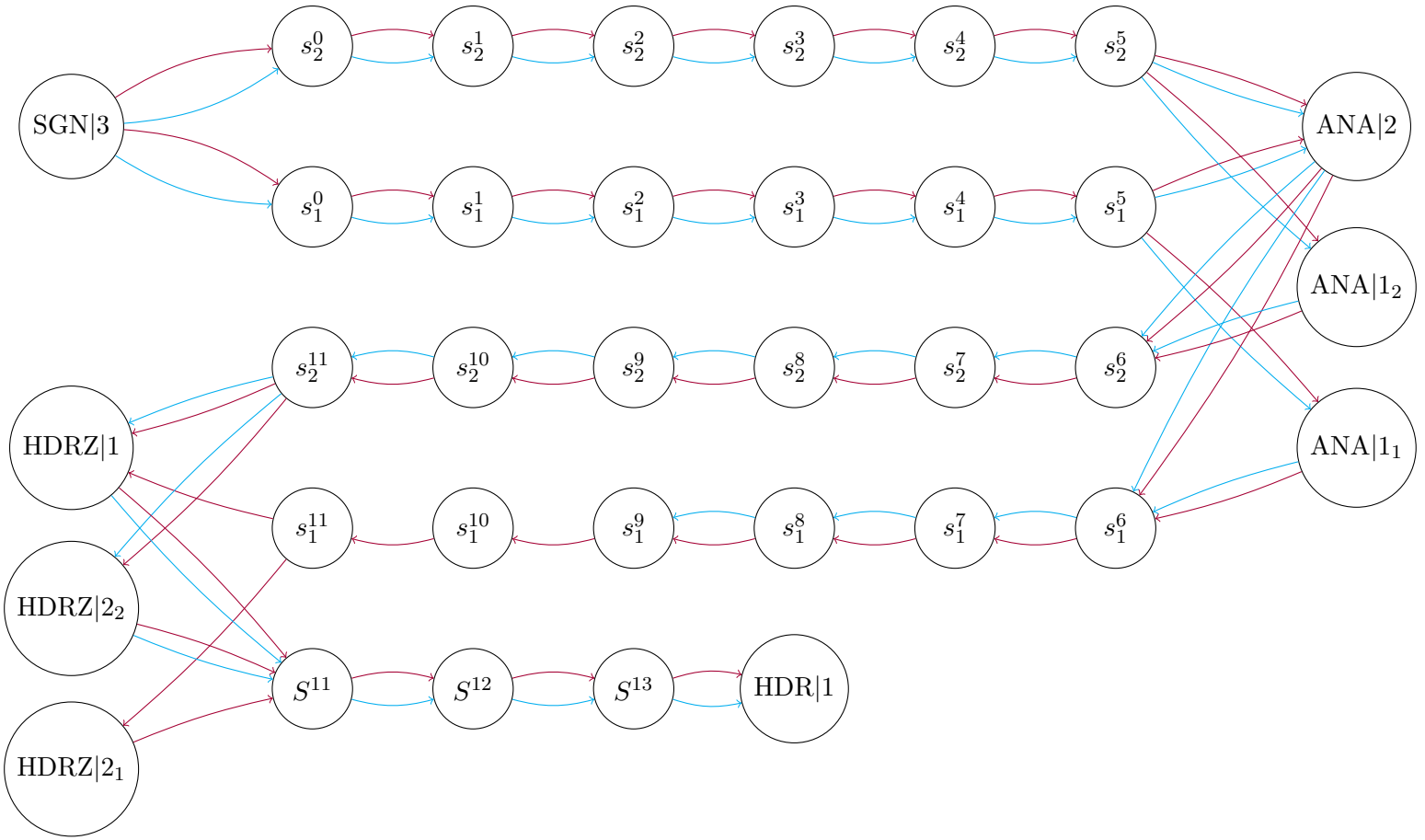


Figure B.1: Search graph of FlexSIPP of the scenario of Den Helder, burgundy edges represent edges that use the flexibility of an agent. Only showing possible routes to platform 1 of Den Helder.