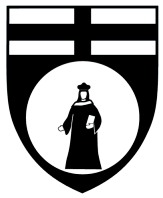
UNIVERSITÀ DEGLI STUDI DI GENOVA
SCUOLA POLITECNICA
DIBRIS
DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI



Università di Genova

MASTER'S DEGREE IN
COMPUTER ENGINEERING - CURR. ARTIFICIAL INTELLIGENCE AND HUMAN-CENTERED COMPUTING

Academic Year 2021/2022

In-Station Train Dispatching via Artificial Intelligence Techniques: Optimisation, Rescheduling and Visualisation

Candidate
Alessio Formica

Supervisor
Prof. Marco Maratea
co-Supervisor
Dott. Matteo Cardellini

Abstract

The In-Station Train Dispatching problem concerns the computation of a schedule of train movements within the station by respecting imposed constraints and an official timetable. A schedule based on studied evaluations of these movements can ensure an improved use of station components and a more efficient time management, as well as a reduction of delay. This process, in order to ensure reliability, should consider possible service disruptions that may compromise the validity of the executed schedule and timely propose an alternative solution.

The objective of this thesis is to extend an approach to perform in-station dispatching, which is based on artificial intelligence techniques, with an optimisation of the quality of schedule and a rescheduling of it when a service disruption is detected. These extensions, optimisation and rescheduling, use a modelling of the problem described through an AI planning language, and, for each of them, different strategies have been defined. With regard to optimisation, different iterative techniques are presented, while for rescheduling, the focus is the presentation of a technique related to the repair of the schedule that is no longer valid. The implemented strategies are tested through real data from a station in the North-West of Italy.

«Buongiorno e casomai non vi rivedessi,
buon pomeriggio, buonasera e buonanotte »

Contents

Chapter 1

Introduction

1.1 Context and Motivations

Rail transport still plays a key role in the transportation of goods and people for short and long distances. According to [for Mobility and Transport, 2019] rail activity in Europe is high, and it is estimated that by 2050 passenger transport will grow by 42% and freight transport by 60%. The reasons for this growth are to be found in the more positive aspects compared to other transportation modes such as safety [for Railways (EU body or agency), 2022], sustainability, energy consumption, use of space and noise levels. [Givoni et al., 2009].
The increasing usage of rail transport does not only imply a demand for more and faster connections but also a higher reliability. This reliability means punctuality, resistance to unexpected events, minimisation of delays and, in general, in the case of passenger transport, every small improvement for the customer satisfaction.

The solution cannot only be an infrastructural improvement, but it has to also include an improvement in daily operations. It is necessary to introduce and to use the technologies related to Industry 4.0 such as (i) *Artificial Intelligence (AI)*, (ii) *Cloud Computing*, (iii) *Big Data*, (iv) *Internet of Things (IoT)*, (v) *Cybersecurity* [Laiton-Bonadiez et al., 2022] not only for monitoring and controlling the infrastructure, but mainly to automate and to schedule, in an optimal way, the operations.

Stations, due to their structural nature, are critical points in the railway network. The large number of interconnected routes, able to receive numerous trains, obliges the station to meticulously organise its activities. Within it, there are several issues that can be faced. The In-Station Train Dispatching problem, i.e. the problem associated to the schedule of train movements within the station to resolve conflicts between them is certainly crucial. Such conflicts, which produce delays or cancellations on the original timetable, are still managed manually by operators, the dispatchers, whose task is to monitor and intervene on schedules with a very limited set of tools.

Although dispatchers base their choices on predefined rules and experience-based skills, they cannot be used as the only tool to predict the many behaviours of the railway network and to reschedule at a higher level. In fact, if not properly tackled, these conflicts can cause local delays that can propagate over the entire railway network.

The main difficulty of this problem is that it is inserted in a very rapid and variable context. The strategies that are identified for the automatic resolution of this problem must not only take into account the rapidity of the execution time and the quality of the solution, but must also guarantee robustness when the found plan is no longer valid, i.e. ensure a constantly valid assignment even during disruptions such as unavailable infrastructure, external factors or other kind of problems.

1.2 State-of-the-art

The main contributes for the application of an optimisation process on an In-Station Train Dispatching problem refer to [Cardellini et al., 2021b, Cardellini et al., 2021a]. In particular, these works have produced the modelling of the In Station Train Dispatching problem as an AI planning problem using PDDL+ language. The use of the PDDL+ language has allowed to handle the continuous flow of time and the occurrence of events outside the controlled world. In addition, in order to solve complex problems, three main enhancements, implemented on a domain independent planning engine, have been presented: (i) an advanced heuristic, (ii) a time-dependent discretisation of the time and (iii) a series of constraints pruning the search-space. This set of enhancements, in addition to providing a time reduction in solving complex instances, also integrate an implicit search for an improved plan, i.e., a search that favours the extraction of plans based on the desired purposes inserted through the heuristic and constraints.

Different approaches, instead, have been presented for the analysis of possible train-related disruptions and for the definition of models to reschedule a timetable that is no longer valid due to delays, deviations or interruptions. An approach such as [Binder et al., 2017] proposes an Integer Linear Program (ILP) model to reschedule the timetable considering as objectives the passenger satisfaction, the operational costs and the deviation from the undisrupted timetable. A similar approach of an Integer Linear Program (ILP) model to solve the real-time railway timetable rescheduling problems is described in [Veelenturf et al., 2014]. In this analysis, a large network consisting of stations and their connections is considered. It is analysed only the scenario in which tracks are occupied, and it is proposed, as solution, the possibility of re-routing trains to reduce the delay or the number of cancelled trains. Each train service like departure or arrival is represented by an event and the main idea is to minimize a function composed by a weighted sum of the number of cancelled trains and the sum of the delays of all the events from their original scheduled times. Others, instead, such as [Rodrigo Acuna-Agost and Guey, 2011] proposes a MIP formulation applicable to the local context of a single station to tackle the problem of finding a new train schedule after a case of disruptions. The proposed approaches use

the original, no longer valid, scheduling to create the new valid one. More specifically, one strategy, right-shift rescheduling, fixes certain integer variables to maintain a certain order of trains. Another strategy is to use the original scheduling to produce local branching cuts to add to the MIP model. [Dollevoet et al., 2017], instead, proposes an iterative framework to find a new feasible solution by considering combinations of rescheduling resources such as timetable, crew or rolling stock after the occurrence of a disruption. Most of the cited works focus on rescheduling analysis through a more extended approach of the railway network. Similarly, [Kecman et al., 2013] attempts to analyse possible solutions in the event of propagating delays or deviations from the original timetable at different network level as, for instance, by extending its working inputs to the entire Dutch national railway network, using alternative graph strategies and sophisticated metaheuristics. A particular work has been presented by [Li,] that propose MAPF (Multi-Agent Path Finding) algorithms for planning and replanning applied for a Flatland challenge with the representation of large-scale rail networks, but which can nevertheless be applied to more real-world scenarios.

1.3 Goals and contributions of this thesis

Starting from the formulation of the In-Station Train Dispatching problem and the possibility of solving this problem automatically presented in [Cardellini et al., 2021a, Cardellini et al., 2021b], the objectives of this thesis mainly concern the extension of the problem by considering an optimise computation and the possibility of dealing with criticalities through rescheduling operations.

As a natural consequence, the new problems referred to this analysis will also be modelled using the `PDDL+` [Fox and Long, 2006a, Fox and Long, 2006b], a planning domain description language for modelling mixed discrete-continuous planning domains.

For a correct identification of an optimisation strategy, a reference metric has been presented for the comparison of valid plans for the same instance. In addition, a technique for the individuation of the reference metric in ideal conditions of the problem has been shown. This ideal reference metric is used for the evaluation of the quality of a plan, a useful value for the comparison of plans of different instances. Based on the reference metric, five different iterative optimisation strategies have been defined. Among these, the main ones are three and they are implemented through (i) a specific constraint on the general metric (ii) a series of constraints based on the general metric and on the determination of future events and (iii) a series of specific constraints for each train, including a constraint defined through a process of selecting the train that is estimated to need more optimisation.

Definitions and encodings of some critical disruptions related to the railway environment that may occur after the elaboration of a plan are introduced. The disruption scenarios concern (i) the non-availability of a piece of track, (ii) the non-availability of a platform and (iii) the insertion

of a new train not considered in the original schedule. Once the possible criticalities have been described, some strategies to correct the plan are defined and implemented. In particular, a plan repair technique, in which information from the original invalid plan is used, is compared with a simple replanning strategy which only try to find in a standard way a new schedule with the invalid conditions removed from the input or the new conditions added.

Domain-specific enhancements to face the optimisation and rescheduling extensions are added to an already modified version of a `PDDL+` planner already capable of solving large and complex In-Station Train Dispatching problems. Enhancements concern the addition of more controls in the planning state for a more efficient search and the identification of a different planner structure for the execution of rescheduling problems not supported by the original structure.

In order to simulate and visualise the new operations, a web application is implemented. It provides the possibility of visualising different components of the station (such as itineraries or tracks) and the possibility of visualising an extracted schedule over time. In addition, the main rescheduling and optimisation operations that are presented in this study can be simulated and visualised.

1.4 Structure of this thesis

Chapter 2 provides a formalization of the station concept, and it describes, based on different types of trains, the allowed movements and the final goals for each of them. Then, the dispatching problem is presented through the main conditions it has to face. Finally, optimisation and rescheduling problems are presented as extensions of the dispatching problem.

In Chapter 3, the files provided by Rete Ferroviaria Italiana (RFI), the owner of Italy's railway network, for the station formalization, the problem encoding, and the creation of instances to be used for statistical analyses are described. The second part of the chapter is devoted to a review of the encoding of the In-Station Train Dispatching problem presented by [Cardellini et al., 2021b] through the PDDL+ (planning Domain Definition Language) planning domain description language. The optimisation and rescheduling problems related to this work use the original presented encoding.

Chapter 4 is reserved for the description of ENHSP [Scala et al., 2016, Scala et al., 2020], the used PDDL+ solver. The first part of this chapter describes the main structure of the solver. Subsequently, the main enhancements introduced by [Cardellini et al., 2021a] to tackle the issue of complexity of the In-Station Train Dispatching problem are reviewed and summarised.

Chapter 5 is entirely focused on the topic of optimising a plan of an In-Station Train Dispatching problem. Initially, a reference metric is discussed and identified to justify the optimisation process of a problem. The chapter continues with the presentation of different optimisation strategies and their implementation. Finally, a comparative analysis through real data is shown among the

different presented strategies in order to define their advantages or disadvantages.

Similarly, the topic of rescheduling applied to a dispatching problem is covered in Chapter 6. Initially, the rescheduling problem is presented and the two main solving strategies are described: replanning and plan repair. The two strategies are then compared on real data through the use of three different possible disruption scenarios: unavailability of a track circuit, unavailability of a platform, and presence of a new train not included in the timetable of the considered instance. Within the chapter a modification of the planner structure and, in particular, of the execution order of actions and events is also presented in order to solve problems with multiple simultaneous events and events that are a consequence of an action at the same instant.

The optimisation operations and specific rescheduling implementations on different scenarios have been included in a visualiser implemented via web application. The features and tools of this web application are described in Chapter 7. The chapter also shows different techniques used to classify the track circuits of a station according to their criticality.

Chapter 8 contains the presentation of other works related to the main topics of this thesis and a collection of different rescheduling strategies applied to general planning problems.

Finally, Chapter 9 describes different possible future works, and it presents the conclusion.

Chapter 2

Problem definition

The main objective of this chapter is to formalise the concept of a railway station and describe the different analysed problems associated with it. Section 2.1 describes the topology of a station through a possible classification and a formalisation. In Section 2.2, the movements allowed to a train within the station are described. Then, different train types are compared and the goals for each of them are presented. Section 2.3 presents the In-Station Train Dispatching problem and also defines its main conditions. Finally, sections 2.4 and 2.5 respectively describe the *optimisation* and *rescheduling* problem as extensions of a dispatching problem. In the last case, some disruption scenarios which could cause the necessity of a rescheduling strategy are presented.

2.1 Topology of a railway station

A railway station is defined as a service location, i.e. a place where the function of regulating railway traffic takes place. It is used to provide precedence among trains travelling in the same direction and, in the case of a single track, to regulate the movement of trains travelling in the opposite direction. Within the service charter provided by RFI and quoted by [Moscarelli et al., 2017] there is a classification of a station on the basis of different parameters. In particular, a station can be *Platinum*, *Gold*, *Silver* or *Bronze* and the classification is based on daily passenger frequency, station size, commercial offer and intermodality with other means of transport.

The formulation and analyses described in this thesis refer to a gold station, i.e. a medium-sized station. However, their validity extends to stations of smaller capacity, such as the silver and bronze stations.

A railway station can be expressed by a tuple $\mathcal{S} = \langle G, I, P, \mathcal{E}^+, \mathcal{E}^- \rangle$.

- G is an undirected graph in the form $G = \langle S, C \rangle$. The set of nodes S identifies the station segment tracks, i.e. the minimum controllable units, and the set of edges C identifies the connections between them.
- I represents the set of itineraries of a station. Each element of I, $i = \langle s_{m1}, s_{m2}, ..., s_{mn} \rangle \in I$, is a path graph, i.e., a set of connected segment tracks, grouped together by experts, to identify a possible movement in the station. Each itinerary not only defines a possible movement of the train within the station, but also defines the direction of movement. In fact, it consists of an ordered sequence of segment tracks and two specific start and exit-points called flags. Flags are graphical constructs, placed near significant points like traffic lights that regulate the movement between one itinerary and another, and they are used as a reference for movement between two points.
- P represents the set of station platforms for embarking or disembarking passengers. The single platform, $p \in P$, consists of a set of circuit tracks $\{s_i, \ldots, s_j\}$, that identify the position in the station in which trains can access the platform.
- $\mathcal{E}^+ = \{e^+\}$ is the set of entry-points into the station from the outside.
- $\mathcal{E}^- = \{e^-\}$ is the set of exit-points, i.e. points of exit from the station into the rail network outside.

  Entry and exit points are also called portals and, in general, are the points where a train enters or leaves the station. In the formulation adopted, portals are like buffers of infinite size in which trains can wait to enter or leave the station.

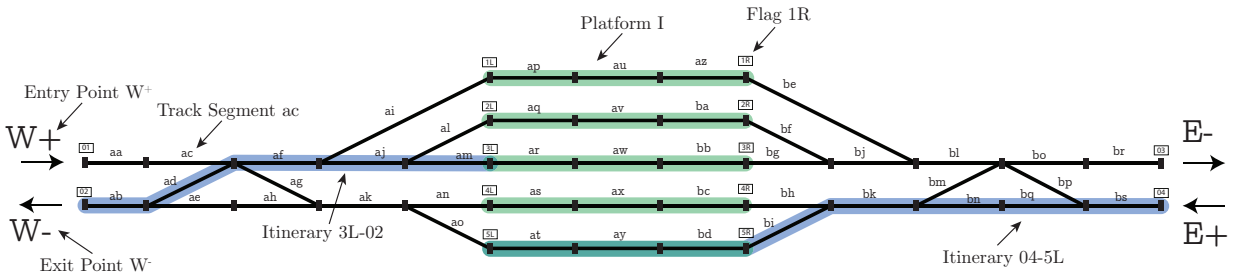


Figure 2.1: Schematic representation of a medium-sized (gold) station. Examples of itineraries are highlighted in a blue colour. One itinerary allows the train to move from the East entry-point ($\mathtt{E}^+$) to the Platform `V`, the other brings a train from the Platform `III` to the West exit-point ($\mathtt{W}^-$)

Figure 2.2 shows a schematic representation of a station. The minimum unit that composes the station is the segment track, and each of these has an identifier. In this representation, there are a pair of entry-points and exit-points on each side. The rectangles represent the flags, and inside them there are their identifiers. Platforms, which are highlighted in green in this figure, are represented as a set of segments tracks and are delimited by two flags. They are uniquely identified

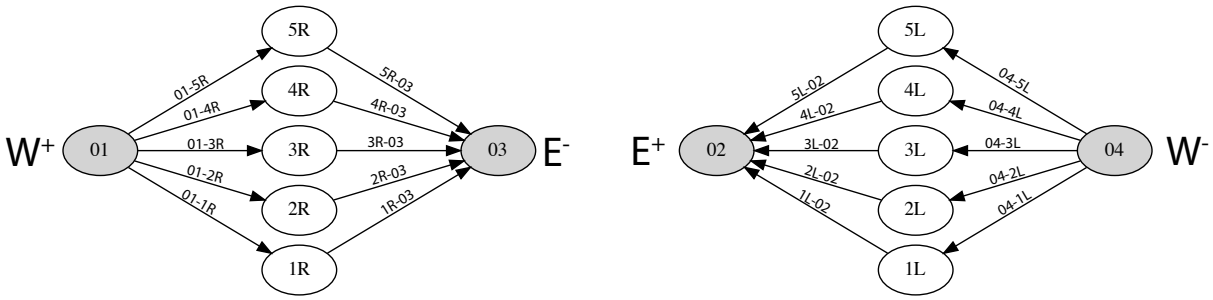


Figure 2.2: Graph of the itineraries. Nodes are flags and the connecting oriented edge are itineraries. Gray edges coincide with portals of the station.

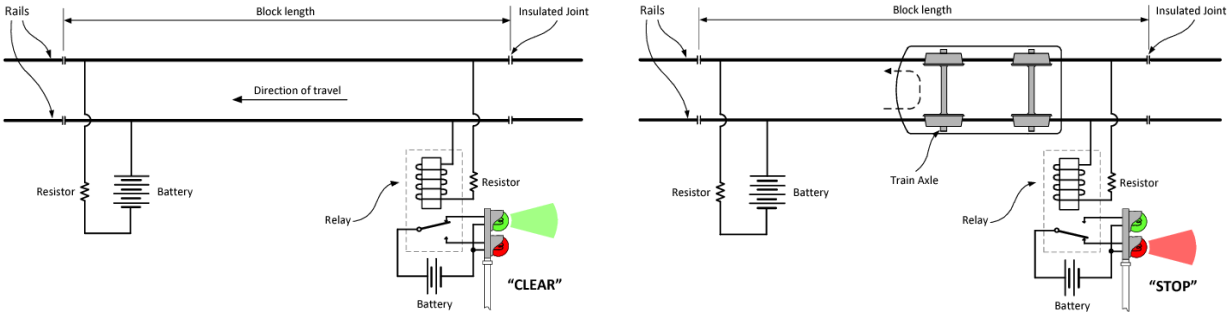
with a roman number (e.g. `VI`) starting with the platform at the top of the representation. Depending on the orientation of the representation, the flags relative to the platforms are indicated by the identified number of the platform combined with their relative position. For example, the right flag of the platform will be identified with `1R`, while the left flag will be identified with `1L`. Within the representation, two itineraries are highlighted in blue. The identifier of the itinerary is formed by joining the flag identifiers at the extremes in the order of the taken direction. For example, itinerary `3L-02` is the path connecting flag `3L` and flag `02` and will be used to continue, after a stop, to the west exit of the station. Similarly, itinerary `04-5L` will start its path from flag `04` to move on platform `V`.

Figure 2.2 shows a different and higher level of itineraries representation within a station using a directed graph. The nodes in the graph represent flags, while the edges represent the itineraries connecting the two points. The nodes in grey are the flags associated with the portals, i.e. the entrances and exits of the station. In the representation, the itinerary is the minimum unit of movement and even if it shares a subset of circuit tracks, it should be seen as a disjointed path with respect to the others. The abstractness of the graph is useful to verify the directionality of the paths and all possible itinerary movements from a portal to another one. In addition, the itinerary graph is acyclic, therefore a reverse movement is not allowed to exit a portal which is in the same direction as the entrance.

2.2 Movements of trains inside the station

In a station representation, a track segment can only be occupied by one train at a time. At the level of railway components, segment tracks are a visual representation of the physical components known as *track circuits*, i.e. electrical circuits whose purpose is to signal the presence of trains on the track.

Track circuits are blocks of different lengths that partition the railway. These segments have



(a) Track circuit off

(b) Track circuit on

Figure 2.3: A schema of a DC track circuit as shown in [Scalise, 2014]. The figure on the left depicts a track segment and the flow of current when no train is running on it. When a train approaches the block, its wheels and axles connect the two running rails together shorting the battery, thus signalling the presence of a train.

a train detection and train speed control function. In detail, quoting [Scalise, 2014], the track circuit consists of a block section that is enclosed on both sides by insulated rail joints. The latter have the task of electrically isolating the track circuits close to each other. At one edge of the block section there is a signal source (a battery in the case of DC or a transmitter in the case of AC) connected to the rails and on the other side of the block there is the receiver (a relay). The track circuit has two possible states. A track circuit is *unoccupied* when no train is present, and thus the direct current supplied by the signal source energises the relay through the running rails. In this case, the green signal light is turned on. When a train approaches the block and its wheels and axles connect the two running rails together, shorting the battery and thereby reducing to zero the current through the relay, a block *occupied* state is generated, the green light is switched off and the red light is switched on. A representation of the two possible states is shown in the Figure 2.3.

Another important physical component for the movement of a train in the station is the *switch*, i.e. a device that allows one or more vehicles to move from a track to another connected one, enabling the independent movement of two or more train sets. In particular, the switch can have two positions: *normal* and *reverse* depending on the track circuit to which it interfaces. The position of the switch determines the continuation of the train position. In one case the train will continue on its route, in the other case the switch will allow the train to move on a track parallel to the initial one.

At the base of station movements, a train, for safety reasons, can only occupy an itinerary if it is not occupied by any other train. Once occupied, all circuit tracks linked to the itinerary are switched into an occupied status. While the train is moving on the itinerary, the freed segment tracks are released. The reason is to give the possibility for the other trains to occupy other itineraries with the same subset of segment tracks.

For an initial description of the possible movements of a train within the station, the graphical representation of the indirect station graph in Figure 2.1 can be used. It shows the connections between track segments within the station. Although, such representation gives the opportunity to define through adjacent segments possible train movements, it is not able to represent cases where movement is forbidden. For example, movement through the segment tracks `aa`, `ac` and `ad` is forbidden even if they are adjacent to each other. The reason is that the angle between `ac` and ad is too tight. A train coming from segment `aa` and `ac` will instead be able to move either on segment `af` or `ag` depending on the chosen itinerary and consequently on the normal or reverse position of the switch. The itineraries `01-1R`, `01-2R` or `01-3R` impose to move through the track segment `af`. Instead, using itineraries `01-4R` or `01-5R` the next segment will be `ag`.

The itinerary graph in Figure 2.2, on the other hand, does not show the possible movements between train segments but gives an indication of all possible directions and itinerary connections between entry and exit points.

Within a station, four categories of trains can be identified:

- *Origin*: the train has its departure within the station. The train departs from a platform and leaves the station via a station exit-point.
- *Destination*: the train ends its journey inside the station. After entering the station through an entry-point, the train moves on a platform to end its journey.
- *Transit*: the train passes through the entire station without stopping at any platform. After entering the station through an entry-point, the train proceeds directly to an exit-point to leave the station.
- *Stop*: the train passes through the entire station making a stop. After entering the station through an entry-point, the train stops at a platform to board or alight passengers and exits the station via an exit-point.

Each train is identified by a unique numeric identifier. This identifier is used every day to identify the same train and its related route.

When a destination train stops on a platform, after waiting for some time, it can depart for a new journey. In this case, the new train will be identified in the timetable as an origin train and its identifier will change.

The information concerning the arrival and departure times of a train of *origin*, *destination*, and *stop* types is collected within a schedule called a *timetable*. In particular, the arrival time of a train is defined as the time at which the train stops at a platform to let passengers disembark or embark. The departure time, on the other hand, identifies the time at which the train leaves the

platform. These times are defined on the timetable as *scheduled times*. However, there can be cases where the timetable is not respected and therefore arrival and departure times change.

In general, the departure time of a train must respect the following equation:

$$T_d = max\{T_{dt}, T_a + T_{min}\} \tag{2.1}$$

where T_{dt} is the departure time scheduled in the timetable, T_a is the arrival time of the train at the platform and T_{min} is the minimum time the train has to remain at the platform to embark or disembark passengers.

2.3 In-Station Train Dispatching

The In-Station Train Dispatching problem is defined as assigning a path for each train that can respect a timetable as much as possible and follow the rules of train movement within the station. Each problem is defined by a specific station, a list of trains that require a path assignment and their position within the station.

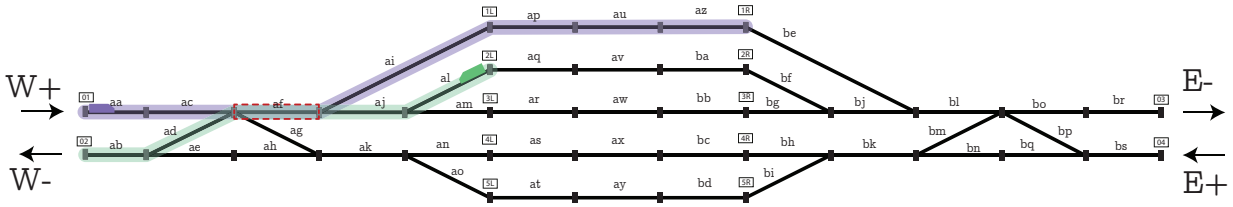


Figure 2.4: Example of incompatible itineraries. The itineraries `01-1R` and `2L-02` are incompatible because they share the track segment `af`.

To ensure correct train movements and a complete respect for the rules within the station, an automatic planner has to deal with the following issues:

1. Each train must be assigned a path to move in the station from their origin to their destination. The origin point, destination point and specific path constraints are defined by the train type. In particular, an origin point of an *origin* train is the platform and the destination point is an exit-point of the station. Instead, a *destination* train has its origin point at an entry-point and a final location on a platform. *Transit* and *stop* trains must move between an entry-point and an exit-point, stopping on a platform if the type is *stop*.

2. When a new itinerary is reserved for a train, all its segment tracks are blocked to prevent another train from occupying them. Therefore, since a track segment can be occupied by one train at a time, when there are two trains that want to move on two itineraries, that

share a segment track this movement, must be managed appropriately. If two itineraries share at least one segment track, they are called *incompatible itineraries*. The figure 2.4 shows two incompatible itineraries: `01-1R` and `2L-02`. The two itineraries share the segment track `af` and then, one train will have to wait until the other finishes its itinerary.

3. Due to the constraint defined in the previous point, the presence inside the station of a small set of trains would quickly paralyse many of the itineraries. This is because, generally, many itineraries share a small set of track segments. To face this issue, times are identified to ensure that a train on an incompatible itinerary can move safely. For example, in Figure 2.4, the second train that moves, does not have to wait for the first train to complete its itinerary, despite being on an incompatible itinerary. It is sufficient for the first train to leave the track segment, shared between the two itineraries, `af` to allow the second train to move.

4. The provided times in the timetable must be respected. In particular, a train cannot, after stopping to embark and disembark people, leave the platform at an earlier time than the time defined in the equation 2.1

2.4 Optimisation

Plan optimisation refers to the extraction of a plan that minimises or maximises one or a combination of metrics. An automatic resolution of the In-Station Train Dispatching problem provides a valid plan that respects determined constraints. However, no particular reference metrics and an explicit process to find an optimal or close to an optimal solution are defined. For example, in a train dispatching problem, the optimisation of a metric that refers to the in-station times of a train could provide a mitigation of accumulated delays and consequently a better efficiency of the station components.

The analysis of this problem requires the study of the following aspects:

1. Definition of a metric whose minimisation or maximisation could represent an improvement in the quality of the solution.

2. Definition of an optimisation strategy that, having defined a problem and a metric to be optimised, produces the optimal solution.

2.5 Rescheduling

The rescheduling problem is defined as a problem of updating the solution of the In-Station Train Dispatching problem due to changes in initial conditions.

The implementation of an efficient and fast rescheduling strategy contributes to ensuring a constant service even in the case of problems and to minimising possible delays caused by timetable changes.

In our analysis, the rescheduling problem is discussed on the basis of three possible changes in initial conditions due to the following disruption scenarios:

1. A first scenario concerns the no longer availability of a segment track within the station due to maintenance work or incidents (i.e. an object falling on the rails causing the track circuit to always signal an occupation). This type of disruption implies the inability of trains to occupy the segment track and, consequently, all itineraries composed by it.

2. A second scenario concerns the unavailability of a platform within the station. The following problem represents a more general case than the previous one, since the unavailability of a platform implies the unavailability of the associated track segments. In this case, the disruption prevent trains from stopping at the unavailable platform to embark/disembark passengers and moving through the related segment tracks.

3. The last scenario concerns the existence of a train not included in the original timetable used by the In-Station Train Dispatching problem. In this case, the presence of a new train must be handled quickly in order to not significantly compromise the timetable organisation.

Chapter 3

Data and Encoding review

This chapter presents the data used to produce a solution of the In-Station Train Dispatching problem and a review of the encoding of the PDDL+ model. The data collected and used for the encoding of the In-Station Train Dispatching problem are presented in the Section 3.1. Section 3.3 provides a review of the dispatching problem formulation also used for optimisation and rescheduling problems.

3.1 Data

Real data from a station in the North-West of Italy has been used to evaluate the analysis. Data were provided by Rete Ferroviaria Italiana, a company that owns and manages the italian railway network, and are contained in four files:

- A CAD file with the illustration of the station structure with references to the main components. A graphic representation allows a better identification of possible movements and a visual evaluation of the correctness of a solution. In particular, it has been important for the implementation of the visualiser in the web application described in the Section 7.
- A JSON-like file containing the data of all structural components of the station. In particular, all itineraries (with related information), platforms, flags, portals, and track segments are stored.
- A Flat-File Database with timestamped logs of the trains movements. Data comes from the Rail Traffic Management System (TMS) which has the task of monitoring and collecting train-related data including:

1. Status of a track segment at a given instant: as stated in Section 2.2 the occupancy or non-occupancy of a track segment is determined by the activation or non-activation of the circuit track, an electrical component. Being only related to the track circuit, the log only shows the presence or absence of a train and it can't, consequently, identify the train.
2. Status of an itinerary: the following log shows the time of occupancy, reservation, and release of a specific itinerary. This information is also associated with the unique train identifier.
3. The arrival and departure time of a train on a platform.

- A file containing the timetable for the station with the following information:

1. The type of train (*origin*, *destination*, *transit*, *stop*).
2. The nominal arrival time on the platform of a train.
3. The minimum time for embarking/disembarking passengers of a train before leaving the platform.

3.2 PDDL+ Formulation

Planning Domain Definition Language (in short PDDL) [Mcdermott et al., 1998] is one of the most important languages for solving automatic planning problems. In particular, an automatic planning problem concerns the identification of a sequence of well-defined *actions* in order to reach a goal state from an initial state. An action is defined as a transformation of the state of the world that occurs under certain preconditions. The language used for the formalisation of the analyses problem is PDDL+, a later version of the original PDDL version.

PDDL+, unlike its predecessor PDDL or PDDL1.0, is suitable for dealing with problems that require modelling of time and mixed discrete-continuous domains. The main additions of PDDL+ with respect to PDDL1.0 concern the *event* and *process* constructs to, in the first case, model situations that cannot be controlled by the planner and that occur under certain preconditions, and in the second case, model a continuous change in numeric fluents. For time modelling and for dealing with time-dependent processes the planner is discritised into steps. Each step represents a time at which actions can be performed or an event can be triggered.

A PDDL+ planning domain model D is described by the following tuple:

$$\mathcal{D} : \langle \mathcal{T}, \mathcal{C}, \mathcal{F}, \mathcal{X}, \mathcal{A}, \mathcal{E}, \mathcal{P} \rangle$$

- The set $\mathcal{T}$ (types) contains the types related to the analysed domain.

- The set $\mathcal{C}$ (constants) contains the names of the individual typed objects within the analysed domain.
- $\mathcal{F}$ is a set of propositional fluents which are applied to a specific type of object, or to all objects. These fluents are either true or false at any point in the planning.
- $\mathcal{X}$ is the set of numeric fluents related to a specific type of object, or to all objects. They may assume a specific numerical value at any point in the planning.
- $\mathcal{A}$ (Actions), $\mathcal{E}$ (Events), and $\mathcal{P}$ (Processes) are sets of transition schema. A transition schema is the tuple $\langle \sigma, pre, \mathit{eff} \rangle$ where:
  - σ s a sequence of constants from ($\mathcal{C}$) or typed variables in $\mathcal{T}$ involved in the transition scheme. ,
  - pre is first order formula representing conditions that must be met in order to perform the transition scheme
  - eff is a set of boolean and numeric effects describing the effects of the transition scheme. It implies true or false assignment to a propositional fluent and a numeric expression assignment to a numeric fluent.

A PDDL+ planning problem is composed by a planning domain, the set of typed objects, initial conditions and goals for the individual instances analysed. A planning problem is thus described by the following tuple $\Pi : \langle \mathcal{D}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ where:

- $\mathcal{D}$ is the PDDL+ planning domain model associated with the problem, which, in particular, defines a valid schema of transitions.
- A set $\mathcal{O}$ of typed objects that are in the instance analysed by the problem.
- A set of initial conditions $\mathcal{I}$ defining the propositional fluents which are set to true and the different numeric values set for the numeric fluents at the beginning of the problem.
- $\mathcal{G}$ is a set of first-order formulas of ground propositional and numeric fluents and it represents the set of conditions necessary to achieve the goal.

The PDDL+ planning problem requires that a PDDL+ plan is found by respecting the imposed constraints. A PDDL+ plan is an ordered sequence of actions performed at a specific time instant that, from a set of initial conditions, verifies a goal state. The sequence of actions must be associated with a timestamp that defines the instant of application. Since events and processes are not directly controllable by the planner, they are not included in the plan.

3.3 Encoding

3.3.1 Introduction to the encoding

Most of the actions and events used for the PDDL+ encoding of the dispatching problem are designed for trains of every type. However, there are some requirements of specific train types that have required customised structures. Section 2.2 presented four train types: (i) *stop*, (ii) *origin*, (iii) *destination*, (iv) *transit*.

Figure 3.1 shows a graphic representation of the interaction of the different actions and events. Events are represented by rectangles and actions by squared rectangles. The processes are represented through the lines connecting the various figures.

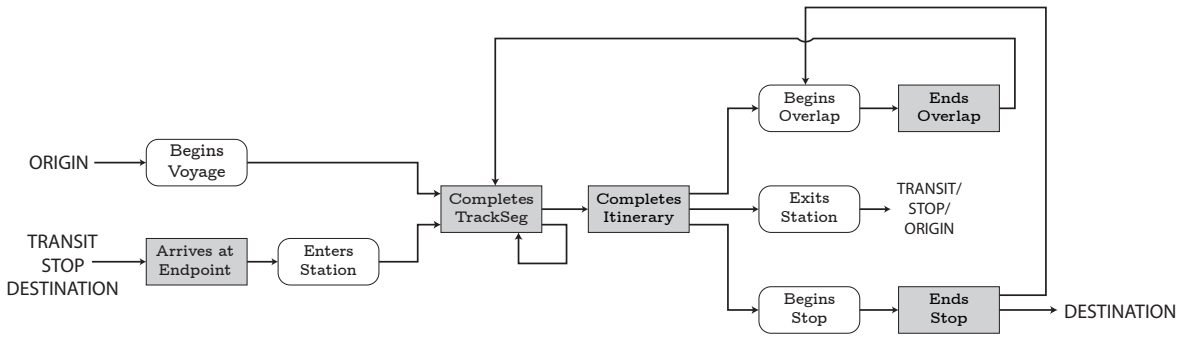


Figure 3.1: A flowchart describing the relationship between actions (squared rectangles) and events (rectangles) regulating the movement of different types of trains within a station.

For movement through itineraries, there are three kinds of actions: `EnterStation`, `BeginsVoyage`, and `BeginsOverlap`. They all have the function of reserving a specific itinerary for the train that will use it to move, and they differ with respect to the time at which the operation is performed. For example, the `BeginsVoyage` operator is only used by a train of type *origin* that wants to start its movement. The `EnterStation` operation is dedicated to trains passing through an entry-point and which want to move within the station. Both operations trigger a process that models the time required to complete the itinerary. During this time, the used and no longer occupied segment tracks are released by triggering the `CompletesTrackSegment` event. If all segment tracks that compose the itinerary on which the train is moving have been released, the planner applies the `CompletesItinerary` event.

The `BeginsOverlap` action, unlike the previous one, is used to generically initiate a movement between two itineraries within the station.

Once a train terminates an itinerary, it may perform different actions: move to another itinerary using the `BeginsOverlap` operation, or if the train is in *transit* or *stop* and has reached an

exit-point, it may exit the station with the `ExitsStation` operation. If the train has reached a platform, the `BeginStop` operation allows the train to perform the stop in order to embark and disembark passengers. The `BeginStop` operation is followed by an `EndStop` event which identifies the moment at which passengers are embarked and disembarked. In the case of a *destination* type train, the train will have completed its journey and a predicate will be triggered to signal its completion of movement. For other train types, the `endStop` event will activate the possibility of moving to another itinerary to continue the journey.

In order to count the time between an action and an event, different processes have been included.

The processes, in the current formulation, identify how much time each train has spent moving on an itinerary, transiting between one itinerary and another (overlapping), staying within the station or at a stop. In addition to the processes just mentioned, a new process has been added to identify the sum of the times in station of all the trains. This value will be used for the optimisation operation described in 5. In the encoding, there is also a process that keeps track of the passing time.

3.3.2 Preprocessor and Grounding

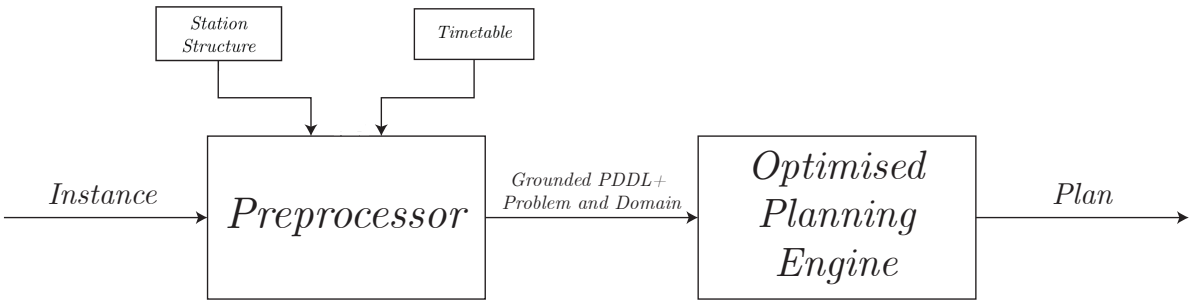


Figure 3.2: Block diagram describing the process of creating a plan. The preprocessor through a specific instance and data related to (i) the structure of the station and (ii) the timetable related to the station and instance, produces a grounded `PDDL+` domain and problem. Through these, the Optimised Planning Engine searches for a valid plan.

The grounding phase of the problem is the first step performed by a solver to assign all constants to variables. In a complex problem such as the analysed, the space to find a valid solution increases significantly as the number of trains increases. Some possible created states, however, are impossible to reach and are consequently useless. The internal grounding of the ENHSP

planner provides a basic static analysis to remove actions and events from the search space whose preconditions will never be satisfied. However, the static analysis does not take into account constraints related to the problem or instance domain.

The preprocessor, shown in figure 3.2 has the role of collecting the dependent information such as station structure and the timetable. It combines them with a single instance of data to directly produce a grounded PDDL+ domain and problem. These will then be solved by the Optimised Planning Engine.

3.3.3 Actions, events, and processes to model the movement of trains

This section will review the PDDL+ constructs used for train movement encoding in the In-Station Train Dispatching problem.

- The event `arrivesAtEntryPoint(t,e+)` (Figure 3.3b) encodes the moment when a train t is approaching to enter the station from the entry-point `e+` and the fluent `time` is equal to or greater than the fluent `arrivalTime(t)`, i.e. the scheduled arrival time at the station. The effect of this event is to activate the predicate `trainHasArrivedAtStation(t)`, which defines that the train is ready to enter the station, and the predicate `trainInAtEntryPoint(t,e+)`.

- When a train has appeared at a specific entry-point e^+, the planner can apply, if the preconditions are met, the action of `entersStation(t,`e^+`,i)` to allow the train `t` to enter the station, moving on itinerary `i`. The action can only be applied when the predicate `trainInAtEntryPoint(t,e+)` is active, i.e. when the train is at the defined entry-point, when the itinerary and all segment tracks that compose it are not occupied and if the itinerary is connected to the entry-point. The effect of the action is to reserve the specified itinerary through the activation of the predicate `trainInItinerary(t,i)` and the blocking of all segment tracks composing the itinerary through the predicate `trackSegBlocked(s)` for each one.

- When a train reserves an itinerary to move on it, the fluent `reservedTime(i)` follows the passing of reservation time of the itinerary i through the process `incrementTimeReservedItinerary(i)` (Figure 3.5b). Each segment track has a specific time of release from track occupation, `segmentLiberationTime(t,s,i)`. When this fluent is reached or exceeded by `reservedTime(i)` then the event `completeTrackSeg(t,s,i)` is triggered, and the segment track is made unoccupied. When the fluent `reservedTime(i)` reaches the `timeToRunItinerary(t,i)`, the predicate `trainHasCompletedItinerary(t,i)`,which defines the completion of the itinerary i by the train t, is activated by the `completesItinerary(t,i)` event.

- The action beginsOverlap(t, i_n, i_m) is used to encode a movement of the train t between one itinerary (i_n) and another one (i_m). The action can only be applied when the predicate trainHasCompletedItinerary(t, i_n) is active and all new itinerary segment tracks are unoccupied.

- The process incrementOverlapTime(t, i_n, i_m) is initialised when a train starts an overlap phase between two itineraries. The aim of this process is to increment the value of fluent timeElapsedOverlapping(t, i_n, i_m) to record the time spent during the operation. The end of the overlap phase of a train occurs when the value of the fluent timeElapsedOverlap(t, i_n, i_m) reaches or exceeds the fluent timeToOverlap, i.e. the time required for overlapping. When this condition is met, the event endsOverlap(t, i_n, i_m) is triggered. As a consequence of the event triggering, the predicate trainInItinerary(t, i_n) is deactivated, indicating the itinerary i_n is no

```
(:action entersStation
  :parameters(
    T1 - train
    W+ - entryPoint
    I01-4R itinerary
  )
  :precondition (and
    (trainIsAtEP T1 W+)
    (not (trainHasExited T1))
    (not (trainHasEntered T1))
    (not (trackSegBlocked aa))
    ...
    (not (trackSegBlocked ba))
    )
  :effect (and
    (not (trainIsAtEP T1 W+))
    (itineraryIsReserved I01-4R)
    (trainInItinerary T1 I01-4R)
    (trainHasEnteredStation T1)
    (trackSegBlocked aa)
    ...
    (trackSegBlocked ba)
  )
)
```

(a) Action entersStation

```
(:process incrementTime
  :parameters()
  :precondition ()
  :effect (and
    (increase time  #t )
  )
)

(:event arrivesAtEntryPoint
  :parameters (
    T1 - train
    W+ - entryPoint
  )
  :precondition (and
    (>= time 242)
    (not (trainHasEnteredStation T1))
    (trainEntersFromEntryPoint T1 W+)
  )
  :effect (and
    (trainIsAtEntryPoint T1 W+)
    (trainHasArrivedAtStation T1)
  )
)
```

(b) Process incrementTime and event arrivesAtEntryPoint

Figure 3.3: The grounded action entersStation which enable a train T1 to enter from an entry-point W^+ through itinerary 01-4R. (right) The process incrementTime which keep track of the flowing of time and the grounded event arrivesAtEntryPoint which signals that a train T1 has reached the exit-point W^+

more occupied.

- The action `beginStop(t,i,p)` is performed to stop a train t, on an itinerary i on a platform p. It follows that the predicates `trainIsStoppingAtStop(t,p)` and `trackSegBlocked(s)` (for each track belonging to the platform) are activated. The time of the stop is modelled through the process `increaseTrainStopTime(t)` (Figure 3.5b) which increases the value of the fluent `trainStopTime(t)` accordingly to the elapsed instants. By rule, a train must wait a minimum time to embark and disembark passengers and this time is identified by the fluent `stopTime(t)`. In addition, the train cannot depart from a platform before a time represented by the fluent `timetableDepartureTime(t)`. Once the values of `trainStopTime(t)` and `time` reach or exceed the limit values respectively, then the event `endStop(t,i,p)` (Figure 3.5b), which indicates that the train has terminated the stop and can resume its movement, is triggered. The predicate `trainHasStopped(t)` is then activated.

- `beginVoyage(t,p,i)` is an action designed for origin trains. It is used to allow the

```
(:event completeTrackSeg
  :parameters(
    T1 - train
    aa - trackSeg
    ac - trackSeg
    I01-4R itinerary
  )
  :precondition (and
    (>= (timeReservedIt I01-4R) 29)
    (trainInItinerary T1 I01-4R)
    (trackSegBlocked aa)
    (trackSegBlocked ac)
  )
  :effect (and
    (not (trackSegBlocked aa))
    (not (trackSegBlocked ac))
  )
)
```

(a) Event `completeTrackSeg`

```
(:process increaseTrainStopTime
  :parameters(T1 - train)
  :precondition (and
    (trainIsStopping T1)
  )
  :effect (and
   (increase (trainStopTime T1) #t)
  )
)

(:process increaseReservedTimeIt
  :parameters(I01-4R - itinerary)
  :precondition (
    itineraryIsReserved I01-4R
  )
  :effect (
    increase (reservedTime I01-4R) #t)
  )
)
```

(b) Processes `increaseTrainStopTime` and `increaseReservedTimeIt`

Figure 3.4: (left) Grounded event `completeTrackSeg` which signals that a train `T1` has finished his run through the track segments `aa` and `ac`. (right) The process `increaseReservedTimeIt` which keeps track of how long a train is reserving itinerary `I01-4R` and the process `increaseTrainStopTime` which counts the time passed for a train `T1` which is stopping at a platform.

departure of a train `t` from a specific platform `p` on an itinerary `i`. The main conditions of application are that the segment tracks of the departure route are all unoccupied and that the departure time is greater than the `timetableDepartureTime(t)` which is the nominal departure time of the train in the timetable.

Each type of train has different initial and goal conditions:

For all trains entering the station, the initial condition is mainly defined by the predicate `train-EntersFromEntryPoint(t,e+)`, which indicates the entry-point of the train.

- *Stop*: The train enters the station and must exit after having performed a stop. To reach its goal, the `trainHasExited(t)` and `trainHasStopped(t)` predicates must both be active.
- *Transit*: The train enters the station and exits without stopping. To reach its goal, the predicate `trainHasExited(t)` must be active.

```
(:action beginStop
  :parameters(
    T1 - train
    I04-3L - itinerary
    SIII - platform
  )
  :precondition (and
    (trainHasCompletedIt T1 I04-3L)
    (not (stopIsOccupied SIII))
    (not (trainIsStopping T1))
  )
  :effect (and
    (trainIsStoppingAtStop T1 SIII)
    (trainIsStopping T1)
    (assign (trainStopTime T1) 0 )
    (stopIsOccupied SIII)
    (not (itineraryIsReserved I04-3L))
    (not (trackSegBlocked aa))
    ...
    (not (trackSegBlocked am))
  )
)
```

(a) Action `beginStop`

```
(:event endStop
  :parameters(
    T1 - train
    S_III - platform
  )
  :precondition (and
    (>= (trainStopTime T1) 300)
    (>= time 421)
    (trainIsStoppingAtStop T1 SIII)
    (stopIsOccupied SIII)
  )
  :effect (and
    (not (trainIsStoppingAtStop T1 SIII))
    (not (trainIsStopping T1))
    (trainHasStoppedAtStop T1 SIII)
    (trainHasStopped T1)
    (not (stopIsOccupied SIII))
  )
)
```

(b) Process `increaseTrainStopTime` and event `endStop`

Figure 3.5: (left) The grounded action `beginStop` which allows the train to stop at platform `SIII` coming from itinerary `I04-3L`. (right) Ground event `endStop` which, when the time has come, signals that the stop has come to an end.

- *Destination*: The train enters the station and stop its journey on a platform. To reach its goal, the predicate `trainHasStopped(t)` must be active.
- *Origin*: The train departs from a platform and must exit the station. The predicates `trainIsStoppingAtStop(t,p)` and `trainHasStopped(t)` are activated as initial conditions, and the predicate `trainHasExited(t)` must be active to reach its goal.

Chapter 4

PDDL Solving

The problems of In-Station Train Dispatching, optimisation and rescheduling has been solved using the open-source ENHSP, a heuristic forward planner [Scala et al., 2016, Scala et al., 2020] with some modifications to adapt it to the analysed domain.

In Section 4.1 it is first described the main structure used by the planner to search a solution. Then, in Section 4.2 domain-specific enhancements presented in [Cardellini et al., 2021a] and required to solve large and complex `PDDL+` In-Station Train Dispatching problems are briefly reported.

4.1 Planning as search

The most natural way to describe a search problem is to use the concept of a search tree. In a search tree, the root node represents the initial state and the intermediate nodes are equivalent to a specific state defined by the state space of the problem. Links represent an applicable action and they change the node state. When an action can no longer be applied to a node, then it is equivalent to a leaf node. Within the search tree, the concepts of event and process are not modelled because these are not directly controllable by the planner.

The Figure 4.1 shows an example of a search tree related to the In-Station Train Dispatching problem.

At the implementation level, a search strategy starts from the initial state and it expands the search tree each time by adding in a list (called `frontier`) a state produced by the application of an action. At each iteration, a new state is chosen from the list, and it is analysed. The process is repeated in loops until a goal state is found. The found path between the initial and final state represents a valid plan.

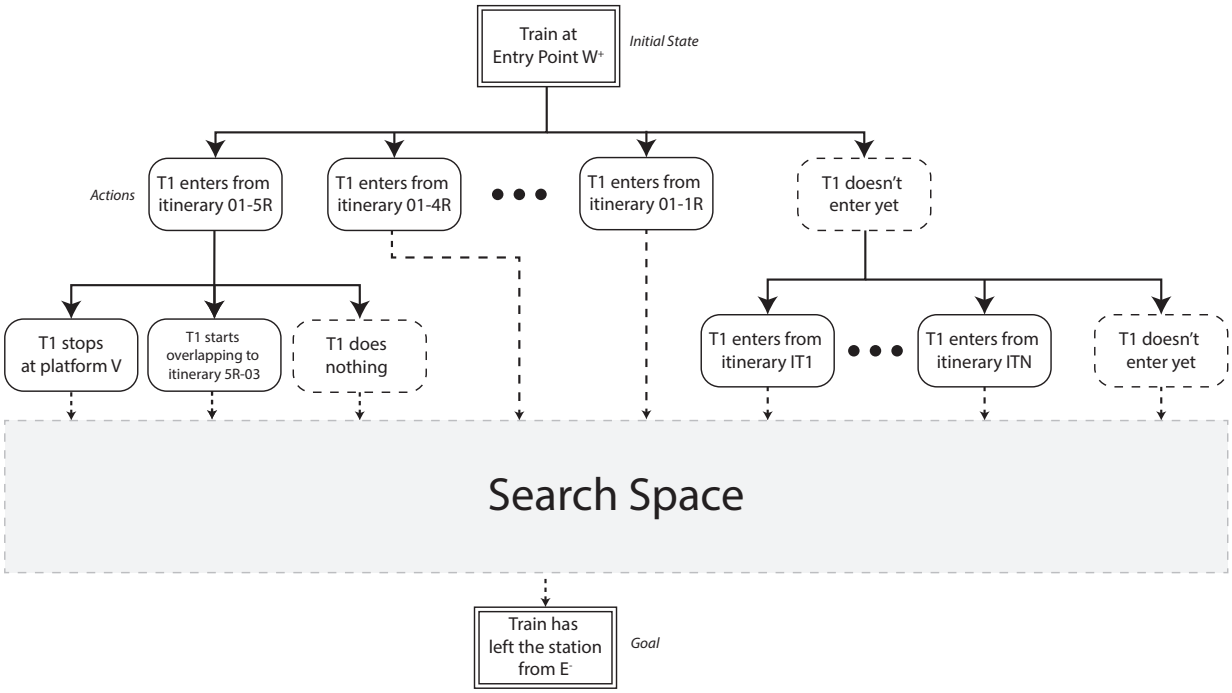


Figure 4.1: An example of a search tree of a dispatching problem in which a train T1 enters from the entry-point $\mathtt{W}^+$ and has to exit from the exit-point $\mathtt{E}^-$. Rounded rectangles represent possible states following an action. States dashed border are a visual representation of the result of applying the *no-op* action in which no action is chosen.

Algorithm 1 shows the pseudocode for the FINDPLAN() procedure, which describes the structure to manage a PDDL+ problem by ENHSP.

Initially, a $node$, representing the initial state of the problem, is added as the first element within the $frontier$. The CHOOSE() function defines the search strategy and returns the node chosen from the frontier for the analysis. The node extracted from the frontier has its state changed by initially applying processes (in $\mathcal{D}.\mathcal{P}$) that satisfy the preconditions through the APPLYPROCESSES() function and subsequently an event (in $\mathcal{D}.\mathcal{E}$) that meets the preconditions through the APPLYEVENTS() function. The function APPLYPROCESSES() is initially called to update all numeric functions by replacing `#t`, the time duration of the processes, with the value of current discretisation. Next, the APPLYEVENTS() function is called and, based on the changes in the previous function, applies an event that meets the preconditions. An action, extracted from the set of the APPLICABLEACTIONS(), is applied through the function APPLYACTION() not directly on the state of the node but on its expansion, so that a new state $expNode$ is created. The APPLYACTION() is executed on all applicable actions. A special action $NoOp$, that has no effect on the node, is added to the list of applicable actions. This action is used when it is more convenient to wait for an event to be triggered instead of executing an action. The EXTRACTPLAN()

Algorithm 1 Algorithm for finding a PDDL+ plan

Input: Domain $\mathcal{D}$, Problem Π and Discretization Step δ
Output: A plan that leads from the initial state to the goal state or $\emptyset$ if no viable plan can be found
function FINDPLAN($\mathcal{D}, \Pi, \delta$)
    $Frontier \leftarrow$ COLLECTION()
    ADD($Frontier, \Pi.\mathcal{I}$)
    while ISNOTEMPTY($Frontier$) **do**
        $node \leftarrow$ CHOOSE($Frontier$)
        APPLYPROCESSES($node, \delta, \mathcal{D}.\mathcal{P}$)
        APPLYEVENTS($node, \mathcal{D}.\mathcal{E}$)
        if STATE($node$) $\models \Pi.\mathcal{G}$ **then**
            return EXTRACTPLAN($node$)
        for all $action \in$ APPLICABLEACTIONS($node, \mathcal{D}.\mathcal{A}$) $\cup \{NoOp\}$ **do**
            $expNode \leftarrow$ APPLYACTION($node, action$)
            ADD($Frontier, expNode$)
    return $\emptyset$

function is executed if a node, that entails the goal, is found. In that case, the search algorithm terminates.

4.2 Improvements

For a better and faster resolution of the described In-Station Train Dispatching problems, some modifications, or additions, to the original planner were necessary. In particular, some added enhancements exploit the features of the domain to improve the search.

4.2.1 Domain-Specific Improvements Review

To deal with the large problems of In-Station Train Dispatching, enhancements presented in [Cardellini et al., 2021a] have been included within the ENHSP solver.

Due to their importance also for the resolution of rescheduling and optimisation problems, they are briefly resumed here:

- The presented PDDL+ model and used for the In-Station Train Dispatching problem is characterised by a ordered sequences of events and actions that mainly depends on the

type of train. This feature gives the possibility to know in advance when an event will be triggered on the basis of an action. As a result, the definition of a discretisation step, required to manage continuous time-dependent processes, may not be fixed but dynamic. The discretisation step can thus be defined on the basis of the action we are performing, since it is certain that at each intermediate step the state of the world will not change. An *Adaptive Delta Queue* has been designed to implement this behaviour. Initially, the queue is initialised with the instants, already defined, at which trains enter the station or depart from a platform. Subsequently, after the execution of any action, all time instants of the events that will be triggered by the action just performed are added to the queue corresponding to the current state. In each state, the next discretisation step is determined by extracting the time of the next event from the corresponding delta queue.

- A traditional A* is used as search strategy. The cost of a search state z is $f(z) = g(z)+h(z)$ and it is composed of $g(z)$, corresponding to the cost of reaching the state z, and $h(z)$, a heuristic estimation of reaching a goal state from the state z. In the strategy adopted for In-Station Train Dispatching problem, the cost of reaching state z is related to the amount of time spent from the initial state to state z. Instead, the domain-specific heuristic is calculated according to the following equation:

$$h(z) = \sum_{t \in T(z)} \rho_t(z) + \pi_t(z) \tag{4.1}$$

  For each train belonging to $T(z)$, i.e. a train that has not yet reached its final destination, the time required to reach a goal state from the current position, $\rho_t(z)$, and a penalty value for each goal not yet met in state z, $\pi_t(z)$, are added together. The heuristic value $h(z)$ is calculated from the sum of all these values.
  This heuristic value also provides the possibility of searching for a valid plan by initially looking for states that have a lower estimated time to complete the objectives. Hence, if time is considered as a metric of evaluation, it is possible to state that this improvement allows to find a plan that is already partially optimised.

- Within the planner, a series of constraints related to the elapsed time of each train have been added for different situations. The idea is to compute a priori maximum times for trains in order to avoid situations in which they stay in the station for an excessive amount of time. At the search level, the constraints are useful to prevent the search tree from being expanded with unnecessary states. In particular, the following constraints were inserted:

$$\texttt{stayInStationTime}(t) < MaxStayInStation \tag{4.2}$$

$$\texttt{fromArrivalTime}(t) < MaxFromArrival \tag{4.3}$$

$$\texttt{stoppingTime}(t) < MaxStoppingTime \tag{4.4}$$

Equation 4.2 constraints the train to stay in the station no more than an a priori calculated maximum time. Similarly, equation 5.3 and 4.4 are related to the time the train arrives at the station and the time spent at a stop.

Chapter 5

Optimisation

In this chapter, the necessary conditions for the creation of an optimisation process are described and different optimisation strategies are compared through extracted results from an analysis based on real data.

Section 5.1 describes the problem and the objectives of an optimisation process. In Section 5.2 we first discuss possible metrics for the In-station Train Dispatching problem and how these can be compared in different scenarios. Then, different possible optimisation strategies are presented to find, through new planning, a better result. In Section 5.3, results of a comparison of the different described strategies are shown and analysed.

5.1 Definition of the problem

The first question, after identifying a technique that easily finds a valid plan, is how we can evaluate the quality of it and, successively, how the search can be improved to find a better result.

To deal with the topic of optimisation for the In-Station Train Dispatching problem, it is necessary to find a metric in which an optimisation process can really lead to an improved result. Then, the main problem is to find efficient strategies to guarantee a correct and, above all, fast optimisation process.

5.2 Encoding and Solving

5.2.1 Metric

An important step in creating an optimisation process is to identify a metric that can represent a good evaluation of improvement. The general idea behind our analysis is to ensure that trains spend as little time as possible in the station by respecting all the constraints and rules concerning train movements in the station.

Three possibilities are:

1. The total time to complete the plan, i.e., the time it takes for the last train to exit the station or to signal the end of its voyage.
2. The sum given by the individual times in station for each train from their entrance. In particular the individual time in station of a train is calculated, (i) if it is a *stop* o *transit* train, from when the PDDL+ action `entersStation` of the train is applied to when it leaves the station, (ii) if it is an *origin* train, from when the PDDL+ action `beginVojage` of the train is applied to when it leaves the station and (iii), if it is a *destination* train, from when the PDDL+ action `entersStation` of the train is applied to when it stops on a platform.
3. The sum given by the individual times in station for each train from they are notified. In this case the individual time in station of a train is calculated, (i) if it is a *stop* o *transit* train, from when the PDDL+ event `arrivesAtEntryPoint` of the train is triggered to when it leaves the station, (ii) if it is an *origin* train, from when the PDDL+ action `beginVojage` of the train is applied to when it leaves the station and (iii), if it is a *destination* train, from when the PDDL+ event `arrivesAtEntryPoint` of the train is triggered to when it stops on a platform. In contrast to the previous option, in this case the timing of stop, transit and destination trains starts from when the trains are notified due to their presence at the exit-point and not from when they are allowed to enter the station via the action `entersStation`.

Regarding the first possibility, the optimisation strategy is likely to be ineffective because only the overall make span is optimised and individual train times are not taken into account.

An example is a scenario with two trains whose arrival time of the second one comes after the exit of the first one. If the second train already follows a path that guarantees to it an optimal time in station while the first has an improvable path, the latter, using this metric, is is already optimal.

Regarding the second and third possibilities, it is useful to mention three important train states used for encoding the In-Station Train Dispatching problem. A train can arrive at a station

entry-point, and from this state it can either enter the station or wait to enter at another time. Successively, when the train passes through the station exit-point, it can exit the station.

It follows, then, that the following inequality always holds true:

$$TimeFromArrival(t) >= TimeFromEnterInStation(t)$$

The problem of using the second possibility is given by the fact that the optimiser, in order to always look for a better time, also exploits the possibility of stopping the train in the virtual buffer outside the station between the entry-point event and the entry station action. The most appropriate solution is to consider the sum of `TimeFromArrival` of a train, i.e., from when the train entry-point event is triggered to when the train exits the station. In this way, the optimiser will consider all time instants since the trains are for the first time notified. The goal of an optimisation strategy based on this metric can be formalised with the following expression:

$$min \sum_{t \in T} TimeFromArrival(t)$$

5.2.2 Quality of a metric

5.2.2.1 Ideal paths

Given a scenario in which there are a variable number of trains, it is possible to calculate the ideal times to solve the problem for each train. In the proposed settings, the term ideal means the best path taken by a train without the obstruction of the other trains or, in other words, the path a train would take if the station was empty.

To find this path, its relative times and metrics, a scenario is analysed and, for each train of that, a new relaxed problem is created that searches for the path of the chosen train after removing the other trains. This problem is then passed through an optimisation process that finds the optimal path for a given train without considering the other trains. For each optimised plan, the individual ideal train metric is then extracted, and the sum, over all the trains, of the individual ideal metrics produces the ideal metric of the scenario. Figure 5.1 shows a representation of the method just described.

The ideal scenario metric, which is derived from the sum of metrics of relaxed solutions, has to be considered only as a reference metric because it does not necessarily correspond to the optimum for that scenario since the interactions between trains are not taken into account.

At the PDDL+ level, the process is performed by creating a domain and problem for each train in the scenario. For each train all information related to trains that are different from the selected

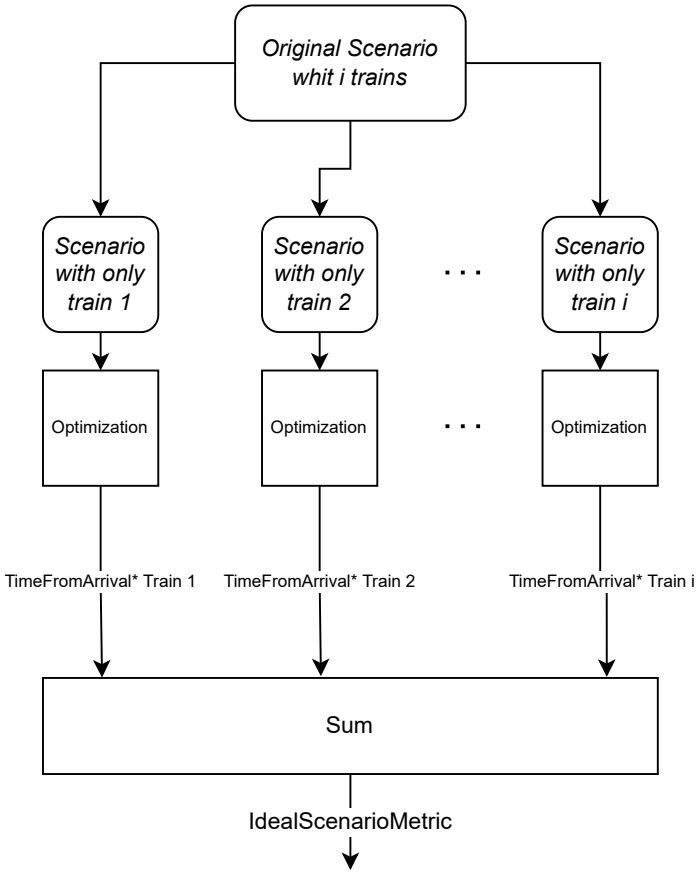


Figure 5.1: A flowchart showing the method to collect the single ideal metric for each train (`TimeFromArrival* Train i`) and the ideal metric for the original scenario.

one are removed from the created files. In particular by removing actions, events and processes from the domain file, initial conditions and goals from the problem file and all references from the heuristics file.

Then, for each problem and domain file, an optimisation process that ensures the optimal metric is performed. In this case, the optimisation process as presented in Section 5.2.3 can be realised by an iterative strategy that optimises the previously found metric at each iteration.

5.2.2.2 Evaluation of the quality

Once the ideal metric for a scenario has been defined, it can be used as a reference to give an evaluation of the quality of a metric found through an optimisation strategy. The quality of the metric is obtained from the ratio of the considered metric and the ideal metric.

$$quality = \frac{metric}{idealMetric}$$

This value is guaranteed to never be less than 1, because the numerator, i.e., the metric considered for the calculation of quality will always be greater than or equal to the denominator i.e., the ideal metric.

The ratio will be equal to 1 when the considered metric is equal to the ideal metric. It is not guaranteed that the quality of the optimal metric for a scenario will be equal to 1 because the ideal metric is calculated on a relaxed problem in which train interactions are not considered, which, instead happens in a standard scenario.

5.2.3 Optimisation strategies

Having defined a metric, several iterative optimisation strategies has been identified.

In our setting, an iterative optimisation strategy is defined as a series of schedules in which at the end of each iteration the domain file, the problem file or the heuristics file is modified in order to optimise the previously found metric. When, during an iteration, the planner does not find any valid plan, the plan found at the previous iteration is *optimal*. The flow of a general iterative strategy is shown in figure 5.2

All the strategies presented have the *anytime* property, i.e., it returns a valid and suboptimal solution even if it is interrupted before its end. The iterative strategy stops when the planner no longer finds a plan. The plan that has the last metric found is called *optimal* for the used strategy.

5.2.4 Optimisation via goals

An iterative strategy in which at each iteration a goal is inserted in the problem file and that condition defines that in the plan the metric must be less than the one previously found. The encoding for the implementation of the strategy within the goal section of a `PDDL+` problem file is shown in Figure 5.3.

5.2.5 Optimisation via preconditions

An iterative strategy in which at each iteration it is added to each precondition of each action and event in the domain file that the metric must be less than the one previously found. In this way, each action and event is constrained to be executed if the metric has not already been

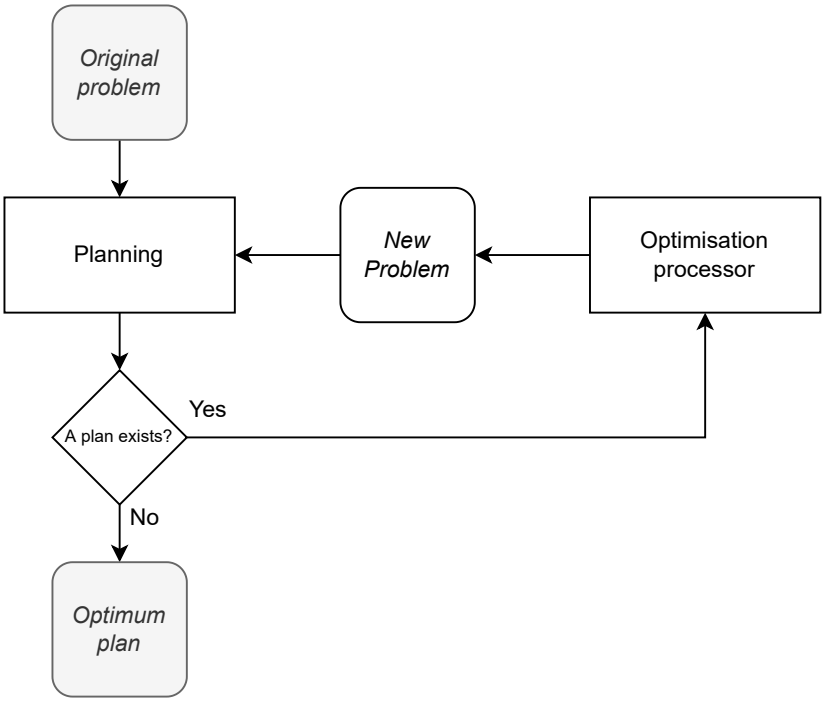


Figure 5.2: A flowchart showing the flow of a general iterative strategy

```
(:goals
    (and
        ...
        ( < (metric) previusMetric)
)
```

Figure 5.3: Encoding scheme for the implementation of the optimisation strategy via goals. Specifically, it is necessary to replace `metric` with the name of the fluent used as reference metric and `previusMetric` with the value of the metric found in the previous iteration.

exceeded. The figure 5.4 shows an example of optimisation via precondition applied to an action in a PDDL+ domain file.

5.2.6 Optimisation via constraint

An iterative strategy that exploits a new constraint within the planner. Before the execution of each action or event, the current metric is compared with the suboptimal metric found in the previous iteration. As the other constraints inserted as planner improvements and described in

```
(:action T11275_exitsStation_I91-07
    :parameters()
    :precondition (and
        (trainHasCompletedItinerary T11275 I91-07)
        (not (trainHasExitedStation T11275))
        (< (metric) (previusMetric))
    )
    :effect (and
        ...
)
```

Figure 5.4: Example of encoding for implementation of optimisation scheme via preconditions. It is necessary to replace `metric` with the name of the fluent used as reference metric and `previusMetric` with the value of the metric found in the previous iteration.

the Section 4.2.1, the check is done before the execution of the action and event in order to speed up the search for the new plan. Once the new plan and its suboptimal metric is found, it is inserted through a pre-processing step within the heuristics file and then used as a reference in the search for a better metric in the next iteration.

The constraint introduced into the planner is represented with the inequality 5.2

$$\texttt{CurrentMetric} < \texttt{Metric} \tag{5.1}$$

5.2.7 Optimisation via dynamic constraints

An iterative process that optimises the previous described strategy through the use of the *calendar queue* structure. The calendar queue is a structure introduced by the enhancements described in Section 4.2.1 and is intended to provide the timing of future events in a given state. Unlike the previous strategy where the constraint was always the same for each action, in this strategy, future events, that are a consequence of the actions the planner performs, are taken into account. This prediction is done using the calendar queue that define at the beginning and during the execution which events come after an action. Also in this strategy, once an improved metric is found, the heuristic file with the new suboptimal metric is updated and a new iteration is performed until the optimal value is found.

In this context, the term dynamic constraint defines this type of constraint that is not fixed but varies according to the action that is being considered.

The pseudocode in Figure 2 shows the schema of an implementation of the strategy adopted to the

Algorithm 2 Algorithm for finding a dynamic constraint

1: **Input:** Node N , Action A and suboptimal metric m
2: **Output:** An upper bound of the current metric calculated through the current action
3: **function** GETDYNAMICCONSTRAINT(N,A,m)
4: $ActionCalendarQueue \leftarrow$ GETNEXTTIMES(A)
5: $CurrentTime \leftarrow N.time$
6: $MaxTime \leftarrow$ GETMAXVALUE($ActionCalendarQueue$)
7: $DeltaTime \leftarrow MaxTime - CurrentTime$
8: $MetricUpperBound \leftarrow m - DeltaTime$
9: **return** $MetricUpperBound$

planner to calculate the upper bound of the metric related to a particular action. Specifically, the GETNEXTTIMES(A) function takes as input the current action and returns, thanks to the calendar queue, a list with the times of events consecutive to the considered action. From this list, only the maximum time instant is extracted through the GETMAXVALUE(ActionCalendarQueue) function. The $DeltaTime$ value, found with the difference between the maximum time extracted from the calendar queue and the $currentTime$ at which the action is being executed, represents the time required to conclude the actions or events caused by the current action. Finally, the upper bound value given by the suboptimal metric found in the previous iteration minus the $DeltaTime$ value is then returned.

The dynamic constraint can be summarized by the following inequality:

```
currentMetric < subOptimalMetric - DeltaTime
```
(5.2)

The greater the prediction of actions or events consecutive to the current action is, the greater the efficacy of the dynamic constraint is.

5.2.8 Optimisation via gradient descent

An iterative process that, unlike the previous ones, works locally on each train. This strategy can be likened to gradient descent because it tries to optimise each time the train that, with respect to other trains, can potentially reduce the total metric.

The name descent gradient, in mathematics, describes an iterative optimisation algorithm designed to find a local minimum of a differentiable function. Conceptually and extended in the context of this study, this strategy presents a similar approach, so it has been given this name.

The pseudocode 3 describes the structure of the algorithm used within the preprocessor to identify the train to be optimised. In addition to the list of trains, the inputs are the ideal times

Algorithm 3 Algorithm for implementing gradient descent algorithm

1: **Input:** CurrentTrainsArrivalTime C , IdealTrainsArrivalTime I and Trains T
2: **Output:** An hash map containing the new list of maxArrivalTimes to use as constraints.
3: **function** PREPROCESSING OF GRADIENTDESCENT(C,I,T)
4: $diffTimes \leftarrow$ HASH()
5: **for all** $train \in T$ **do** State $deltaTime \leftarrow$ GETTIME$(C, train)$ − GETTIME$(I, train)$
6: INSERT$(diffTimes, train, deltaTime)$
7: $SelectedTrain \leftarrow$ GETKEYFROMMAXVALUE$(diffTimes)$
8: $MaxArrivalTimes \leftarrow C$
9: $MaxArrivalTimes.insert(train,$ GETTIME$(C, train) - 1)$
10: **return** $MaxArrivalTimes$

from arrival at station and the arrival times at station of all trains in the current iteration. By operating a train-by-train difference between these two values extracted by the functions GETTIME$(C, train)$ and GETTIME$(I, train)$, the train with the maximum difference, i.e., the train most likely to need optimisation, is selected through the GETKEYFROMMAXVALUE(diffTimes) function. The output represents the new time constraints to be respected for each train. To the selected train is set a maximum time from arrival at the station equal to the previous iteration minus one. The other trains not involved are maintained with the arrival time from the station found in the previous iteration. The output hash map is placed in the heuristics file. It represents the maximum times since the arrival of a train at the station. A specific constraint in the solver is then used to prune the current branch if the condition is not verified.

The constraint used by the planner for this strategy can be formalized as follows:

$$TimeFromArrival(t) < MaxTimeFromArrival(t) \qquad \forall t \in Trains \tag{5.3}$$

5.3 Evaluation

A comparison of the strategies described in the previous section is conducted in order to investigate their properties.

The comparative analysis involved data from different days of a specific railway station in which time windows with different lengths have been extracted in order to identify scenarios defined by a number of trains spanning from 1 to 30 and a timetable to be respected. For each scenario, `PDDL+` files (domain file and problem file) and a heuristics file has been created.

For each optimisation strategy a cut-off of 30 CPU-planning minutes is imposed (considering all iterations) and, at the end, the final metric value is extracted and the corresponding metric quality value, described in the Section 5.2.2, is associated with it. The 30-minute limit does not guarantee

finding an optimal metric, but, particularly for scenarios with many trains, it guarantees a metric that empirically approaches the optimal one. Finally, for scenarios with an identical number of trains, an arithmetic average of the metric quality is performed.

The multiline chart in figure 5.5 shows the results of this analysis. On the x-axis, we have the number of trains contained in a scenario. On the y-axis, we have the values representing the quality of the metrics. Each line represents an adopted optimisation strategy, and the identified values are defined as the arithmetic average of the quality of the metrics of all scenarios with the same number of trains. There are at least three scenarios with the same number of trains. The dotted line represents the average value of the metric quality for the first iteration, which is identical for all strategies. The considered strategies are limited to (i) strategy with the constraint, (ii) strategy using dynamic constraints, (iii) gradient descent strategy, (iv) all three strategies combined.

As expressed in Section 5.2.2, it is not possible to find metric quality values below a value of 1 because quality is calculated on the basis of the ideal metric, which, by definition, cannot be exceeded. The closer the values are to the value of 1, the better the metric is, but it is not certain that the optimum coincides with the ideal metric.

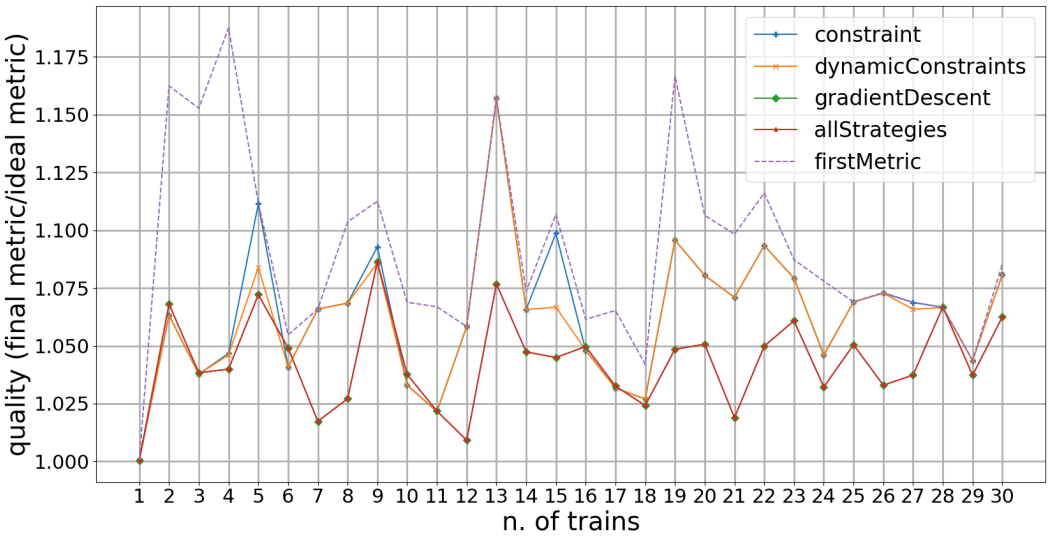


Figure 5.5: Figure shows a multiline chart with the result of the optimisation evaluation. Every line represents a specific strategy and the dotted line represent the quality of the first iteration.

Looking at the graph, the following conclusions can be derived:

1. The trends of the gradient descent strategy and of all individual strategies together are the same. It can therefore be stated that the gradient descent strategy imposes stronger

constraints than the others strategies so that it dominates over them. Applying the three strategies together brings no advantages over using the gradient descent strategy alone.

2. In general, the gradient descent strategy finds a more optimised value of the metric than the other strategies.

3. The application of the dynamic constraints strategy improves the single constraint strategy, finding better metrics on average.

4. Considering the following analysis with all scenarios, compared to the initial metric value, the gradient descent strategy produces better metric values with an average percentage variation of 5%. The percentage variation reaches values such as 14% when scenarios with a smaller number of trains are considered, since these tend to complete the optimisation process.

The significant advantages produced by the gradient descent strategy are contrasted, as already mentioned, by the consideration of the fact that this type of strategy, due to its nature, does not guarantee optimality. However, considering the huge complexity in particular in scenarios with many trains, the strategy offers an excellent trade-off in terms of metric quality and planning time.

To conclude, the optimisation research covered in this chapter had the only intention of proposing some basic strategies and implementations to perform a plan optimisation process of an In-Station Train Dispatching problem. In particular, the basic implementations of the strategy of dynamic constraints and gradient descent, which have already produced excellent results in this form, can certainly also be improved and combined.

Chapter 6

Rescheduling

In this chapter, the rescheduling problem as an extension of the In-Station Train Dispatching problem will be described through the example of three different disruption scenarios. A comparison between rescheduling and plan repair, two different strategies to tackle the problem, will then be presented, and an implementation of these will be used to evaluate their properties through the previously described disruption scenarios.

In Section 6.1, the rescheduling problem is described, and different properties are identified. A description of two different rescheduling strategies are presented in Section 6.2 and, after identifying possible disruption scenarios, their encoding is shown for each of these. A comparative analysis of the two strategies applied to the identified scenarios is presented in Section 6.4

6.1 Definition of the problem

Although railways are a strongly controlled environment and several strict rules are in place that manage movement in and out of the station, it has daily problems originating from different reasons. Many of these issues have an easily solvable local impact, other times the problem propagates significantly, affecting the entire train scheduling.

Hence, if the previously extracted dispatching plan is no longer valid, it becomes necessary to find a new plan that can satisfy the new conditions.

The issues that can lead to the invalidation of a plan can be several. However, the analysis focused on three very common possible disruption scenarios:

1. No longer availability of a segment track within the station. This unavailability can be caused, for example, by maintenance work or incidents that entail the continuous signalling

of the state of occupation by a circuit track. The non-availability of a segment track implies the non-availability of all itineraries that are composed of it.

2. No longer availability of a platform within the station. This non-availability can be expressed as the non-availability of one or more connected circuit tracks. The result is that the platform and the associated circuit tracks cannot be used for the movement of trains and the embarkation and disembarkation of passengers.

3. The existence of a train that has to be managed not included in the original timetable.

The invalidation of the plan then requires a rescheduling strategy that can produce a plan that meets the new conditions.
Two significant properties identified by [Abels et al., 2021] for a rescheduling problem are:

- *Similarity* with the previous plan. This implies that the solution of the new problem deviates as little as possible from the previously found plan. This property is also identified as plan stability, which refers to a measure of the difference that a process induces between an original plan and a new plan. The measure related to plan stability is, in general, not defined. It can be customised in relation to the analysed domain to identify a good way to compare the similarity between two plans. A possible value of plan stability follows the definition given by [Fox et al., 2006]:

  Definition 6.1.1 *Given an original plan, π_0, and a new plan, π_1, the difference between π_0 and π_1, D(π_0, π_1), is the number of actions that appear in π_1 and not in π_0 plus the number of actions that appear in π_0 and not in π_1.*

  According to the same study, preserving plan stability would ensure coherence and consistency of behaviours by making agents more predictable, it would make plan generation faster, and it would guarantee high quality plans because they would share some properties found in the initial plan.

- *Timeliness*, i.e. the ability of the strategy to guarantee quickness in finding the new plan.

The two aforementioned properties are studied and compared in the analysis between the different rescheduling strategies identified.

6.2 Encoding

A rescheduling problem is directly associated with an invalid plan which in an In-Station Train Dispatching problem can be caused by a disruption, i.e. an event that changes the predefined conditions of the railway network.

The analysed disruption scenarios involve the addition of new conditions to the original encoding of the In-Station Train Dispatching problem. The addition of these new conditions is realised by modifying either the problem file or the domain file or the heuristics file. The encoding of rescheduling problems also differs according to the rescheduling strategy considered.

6.2.1 Replanning vs Plan Repair

In this analysis, two general rescheduling strategies are considered and compared: (i) the replanning strategy, or also called planning from scratch, and (ii) a plan repair strategy.

The strategy of replanning (or planning from scratch) simply solves the problem with the new formulation as if it was a new In-Station Train Dispatching problem. Similarity with the previous plan is not guaranteed, since no constraint is defined and no information from the previous solution is used. Similarly, timelessness is not guaranteed since the planning time is closely linked to the solution time of the In-Station Train Dispatching problem.

A plan repair strategy, instead, is defined as a strategy that uses information from the solution of the invalid plan to *guide* the new planning. The idea is to repair the criticalities of the plan that is no longer valid without completely replanning. Therefore, the strategy tries to maintain similarities with the original plan or with information from the search of the previous plan. According to the information used, the strategy can be realised in different ways. In our analysis, the implementation of the strategy directly uses the previously found and no longer valid plan as information. The idea is to keep train actions unaffected by the disruption and giving the affected train the opportunity to find a new schedule.

6.2.2 Action-to-event transformation

The operation of transforming actions into events is a process developed to implement the plan repair strategy used in the analysis. It is divided into two phases:

1. The identification of actions to be transformed into events. In our analysis, the identified actions are those not directly involved in the identified disruption scenario. For example, an action can be identified if it does not involve a particular train or segment track.
2. Anchoring, i.e. fixing the selected actions performed in the no longer valid plan through preconditions and heuristics references also in the plan that is solution of the rescheduling problem.

At the end of the two phases, the selected actions become events and their instant of execution in the planner will be governed by heuristics and preconditions.

In our analysis, as shown in the figure 6.2 anchoring is done directly on all actions of a train in a plan. Thus, once all actions relating to a specific train have been identified, they are anchored, at a specific instant, within the domain file via preconditions. Other actions and events that are not used defined in the domain file, and they are associated with the train involved in the operation, are deleted. Finally, to ensure an efficient and correct search for the new plan, the list of actions and their application times are added to the heuristics file. This is to ensure that the times enters into the calendar queue of the solver. This process forces the planner to execute the new anchored events of the trains affected by the transformation operation. An example of the step performed inside the action in the domain file is shown in the figure 6.1.

```
(: action entersStation
  :parameters(
    T1 - train
    W+ - entryPoint
    I01-4R itinerary
  )
  :precondition (and
    (trainIsAtEP T1 W+)
    (not (trainHasExited T1))
    (not (trainHasEntered T1))
    (not (trackSegBlocked aa))
    ...
    (not (trackSegBlocked ba))
    )
  :effect (and
    (not (trainIsAtEP T1 W+))
    (itineraryIsReserved I01-4R)
    (trainInItinerary T1 I01-4R)
    (trainHasEnteredStation T1)
    (trackSegBlocked aa)
    ...
    (trackSegBlocked ba)
  )
)
```

(a) Action `entersStation`

```
(: event entersStation
  :parameters(
    T1 - train
    W+ - entryPoint
    I01-4R itinerary
  )
  :precondition (and
    (trainIsAtEP T1 W+)
    (not (trainHasExited T1))
    (not (trainHasEntered T1))
    (not (trackSegBlocked aa))
    ...
    (not (trackSegBlocked ba))
    )
   (= time 213)
  :effect (and
    (not (trainIsAtEP T1 W+))
    (itineraryIsReserved I01-4R)
    (trainInItinerary T1 I01-4R)
    (trainHasEnteredStation T1)
    (trackSegBlocked aa)
    ...
    (trackSegBlocked ba)
  )
)
```

(b) Event `entersStation`

Figure 6.1: A standard action (right) An event derived from the transformation of an action. An event derived from the transformation action to event. In the example, in addition to the event tag, the time instant at which the event is to be executed is inserted in the preconditions(left)

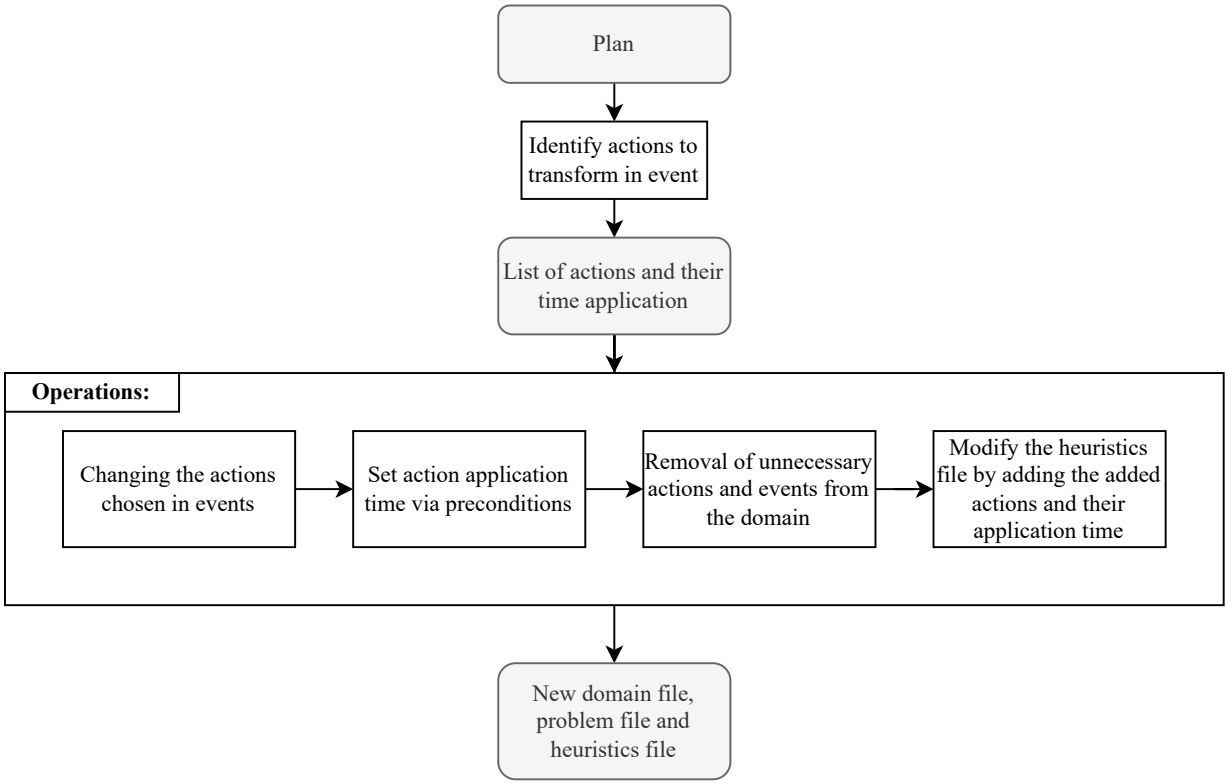


Figure 6.2: A flow chart showing the flow of the Action-to-Event transformation technique. After having identified the actions of the plan to be transformed into events, a series of operations are performed on the problem files. In particular, the actions transformed into events are bound to be applied at a certain instant. After the modifications, the new problem is thus produced.

6.2.3 Disruption scenarios

6.2.3.1 Track circuit unavailable

The rules which control the movement of trains in the station in case of a disruption of a track circuit prevent the movement of trains on it. As a consequence of this condition, all itineraries that include this track circuit cannot be used.

To formalise the condition of unavailability of track circuit, a predicate, `trackSegBlocked(s)`, already contained in the formalisation of the dispatching problem has been used. This condition, included in the initial conditions of the original problem, is a constraint that always remains valid and prevents the use of itineraries containing the unavailable track circuit.

For the resolution via the replanning strategy, the problem formulation just described can be used directly.

For the resolution via the plan repair strategy defined in our analysis, the formulation needs some changes in order to define which trains are affected by the unavailable track circuit.

In this case, a plan repair operation should provide a local resolution of only those trains that are affected by the unavailable track circuit by trying to keep the unaffected trains with the same path found in the original plan.

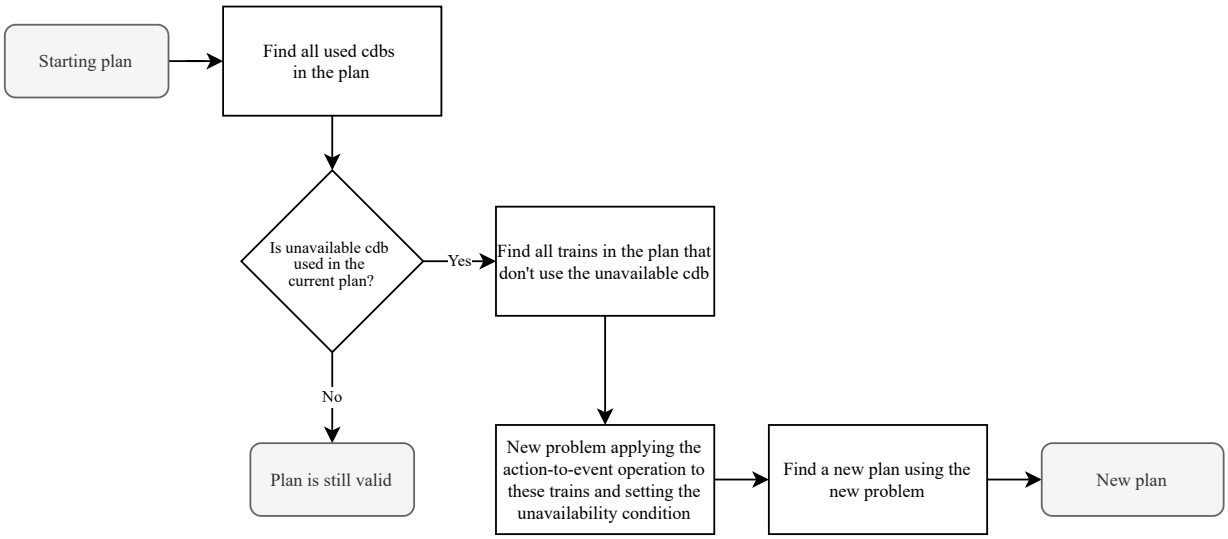


Figure 6.3: A flowchart showing how to repair a plan in case of a track circuit unavailable. The operation of transforming action to events is applied to the trains not affected by the track circuit and a new plan is searched for.

The flowchart in 6.3 shows the plan repair strategy from when a track circuit is unavailable until the plan is solved. Once the unavailability of a track circuit is detected, it is checked whether the plan derived from the original problem contains any train movements which use the track circuit. The strategy is only applied if at least one train uses the track circuit, otherwise the original plan remains valid. If at least one train uses the track circuit, then the affected trains are identified. The action-to-event transformation operation is applied to the remaining trains. Finally, the planner tries to solve the new problem, searching for a valid plan.

6.2.3.2 Platform unavailable

In the current formulation, a platform is identified by one or as a series of interconnected track circuits. The unavailability of a platform produces the unavailability of embarking/disembarking passengers on it, and also the impossibility of moving on the track circuits which compose the platform.

Also in this case we can use two predicates already present in the encoding of dispatching problem in the initial conditions to formalise the problem:

- `trackSegBlocked(s)`
- `stopIsOccupied(s)`

These predicates are included in the initial conditions of the original problem.

This type of disruption can be seen as a similar case to the scenario described previously, since the unavailability of the platform produces the unavailability of the associated track circuit.

One solution for the extraction of a valid new plan is the application of the replanning strategy on the formulation of the new problem. In this way, as in the previous scenario, a plan is searched for as if it were a new dispatching problem.

Otherwise, it is possible to adopt an implementation quite similar to the previous one through the plan repair strategy. Given the similarity with the non-availability of a track circuit, the implementation adopted is the same shown in Figure 6.3. The only difference concerns the application of the operation of transforming actions into events over the trains whose path, in the original plan, doesn't pass through the unavailable platform.

6.2.3.3 New train in the timetable

For the analysis of this disruption case, it is useful to remember that the formulation of the In-Station Train Dispatching problem is related to a static scenario, i.e. a scenario with a number of trains well-defined at the beginning. In this case, the existence of an extra train within a scenario can be encoded by combining the domain file, problem file and heuristic entries of the original scenario with those of the single train.

As in the previous cases, a first possibility for rescheduling can be done via the replanning strategy. It takes the new problem, created by the combination, and searches for a plan as the standard In-Station Train Dispatching problem.

In the case of a plan repair strategy instead, the idea is to keep fixed the paths found by the plan of the original scenario and only search for the path of the new single train. The flowchart in Figure 6.4 shows the key steps for encoding and solving the new problem. Initially, the two problems are combined. The action-to-event transformation operation is then applied to all actions that describe the path of trains that are in the plan of the original scenario. Thus, the solver will try to find a plan that maintains the train paths already in the original plan and that has a valid path for the added train.

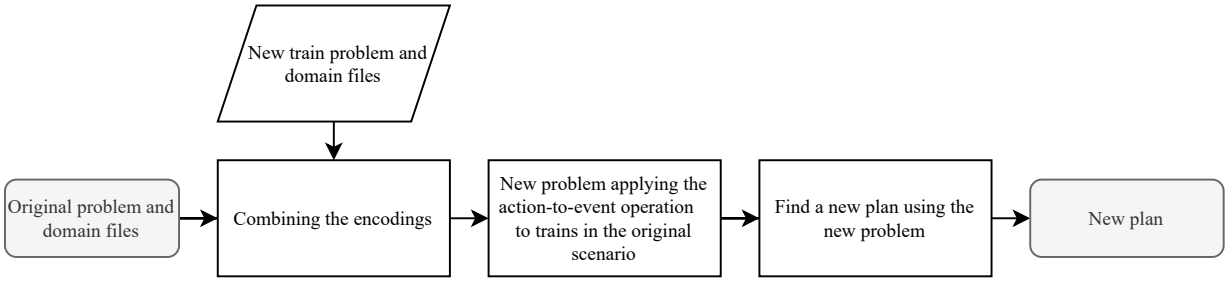


Figure 6.4: A flowchart showing implementation of rescheduling by applying the plan repair strategy. After combining the problems, the action-to-event transformation operation is applied to all trains of the original scenario. The solver is then run on the new problem and the new valid plan is extracted.

6.3 Solving

Both presented rescheduling implementations are based on a different encoding of the initial problem. However, while the encoding of the replanning strategy varies little from the encoding of the original problem, the encoding produced by the plan repair strategy differs significantly.

In the first strategy it can be compared to solving a new In-Station Train Dispatching problem, thus the Domain-Specific Improvements made on the off-the-shelf version of ENHSP are sufficient to guarantee the correctness of the solution.

In the second case, the adopted plan repair strategy produces an encoding with numerous events, even simultaneous, as a result of the operation of transforming actions into events. The consequence is that the off-the-shelf version of ENHSP planner cannot solve problems due to how the ENHSP planning engines solves the `PDDL+` planning problems. In fact, the planner structure is not able to trigger simultaneous events or events after the execution of an action. It is therefore necessary to modify the planner to allow simultaneous events and events that are triggered only after the execution of an action.

6.3.1 Modification of the planner structure

Analysing the original structure of the planner through the pseudocode described by the algorithm 1, we notice that the `ApplyEvents()` function is only executed once, hence, only one event can be applied per iteration. Furthermore, after the `ApplyAction()` function, the planner is no longer allowed to apply a possible event caused by the execution of the action.

This solver structure does not allow handling two possible issues as (i) presence of simultaneous events and (ii) presence of simultaneous and successive events to an action.

Algorithm 4 Algorithm for finding a PDDL+ plan for a rescheduling problem

```
1: Input: Domain 𝒟, Problem Π and Discretization Step δ
2: Output: A plan that leads from the initial state to the goal state or ∅ if no viable plan can be found
3: function FindPlan(𝒟, Π, δ)
4:     Frontier ← Collection()
5:     Add(Frontier, Π.ℐ)
6:     while isNotEmpty(Frontier) do
7:         node ← Choose(Frontier)
8:         ApplyProcesses(node, δ, 𝒟.𝒫)
9:         for all events ∈ ApplicableEvents(node, 𝒟.ℰ) do
10:            ApplyEvents(node, 𝒟.ℰ)
11:            if State(node) ⊨ Π.𝒢 then
12:                return ExtractPlan(node)
13:        for all action ∈ ApplicableActions(node, 𝒟.𝒜) ∪ {NoOp} do
14:            expNode ← ApplyAction(node, action)
15:            for all events ∈ ApplicableEvents(node, 𝒟.ℰ) do
16:                ApplyEvents(node, 𝒟.ℰ)
17:                if State(node) ⊨ Π.𝒢 then
18:                    return ExtractPlan(node)
19:                Add(Frontier, expNode)
20:    return ∅
```

The management of multiple events that are triggered at the same time point has been discussed in [Fox et al., 2005]. The problem relates to the ordering in which the events are executed since if the events are dependent on each other, the execution of a different ordering of these may produce a different final state. Therefore, allowing multiple events to be executed simultaneously may produce an unpredictable state produced in a non-deterministic way. For this reason, the structure of planners and, in particular ENHSP, does not support the execution of multiple events at the same time. However, in our analysis, the application of the operation of transforming actions into events is performed on a plan constructed in a deterministic way through encoding that does not include simultaneous events. Therefore, if they are executed in the same order, events, inserted in specific time instants that previously corresponded to actions, produce an identical effect to the effect they had produced as actions. Thus, the solution remains unique.

Therefore, for its correct execution, modifications have been made to the planner concerning the order in which events and actions are executed.

The modification introduced in the structure of the planner are presented through the pseudocode in the Algorithm 4. The behaviour of all the functions in the pseudocode remains unchanged.

The changes concern the portion where events and actions are applied. Now, all applicable events extracted by the set of `ApplicableEvents()` are applied, and each event changes the state of the node. This addition enables the application of multiple events at the same time.

With regard to the execution of actions, it is checked after each expansion whether new events are applicable. If so, the applied events change the state of the new node. Then, the new node is added to the $frontier$.

This modified version is able to deal with encodings that have the problems previously mentioned. For problems that do not suffer from these problems, the new structure behaves identically to the original planner structure, hence ensuring a correct execution also for standard In-Station Train Dispatching problems.

6.4 Evaluation

In order to evaluate and compare the two different rescheduling approaches described in the previous sections, a comparative analysis is conducted with respect to each defined disruption scenario. The main objectives of the analyses are to verify possible differences between replanning and plan repair strategies with respect to CPU-planning time and quality of the plan found.

In the analyses concerning track circuit unavailability and platform unavailability, different days have been considered and for each of them time windows have been extracted. The aim has been the identification of instances, or scenarios, which span from 1 to 30 trains. For each scenario, `PDDL+` files (domain file and problem file) and a heuristics file have been created.

For both analyses, n sub-scenarios were created for each scenario where n is the number of tracks circuits present in the station (in the first analysis) or the number of platforms in the station (in the second analysis). In the analyses, the scenario in which multiple unconnected track circuits are not available has not been considered since it is a catastrophic scenario and therefore not relevant. The result it that, for the same scenario, all possible combinations in which a single track circuit or platform is not available are considered. Finally, replacement plans are searched for by applying the two rescheduling strategies.

For each sub-scenario in which both strategies have found a valid plan, the percentage variation of CPU-Planning Time from the plan found in the replanning technique to the plan found with the plan repair technique is calculated. An average of the values of each sub-scenario is then calculated to identify an average value of percentage variation of CPU-Time planning for each main scenario.

Figure 6.5 presents a line chart showing the percentage variation of CPU-planning time when the number of trains varies in the case of the unavailability of a track circuit. Similarly, Figure 6.6 shows the same line chart using values for the unavailability of a platform.

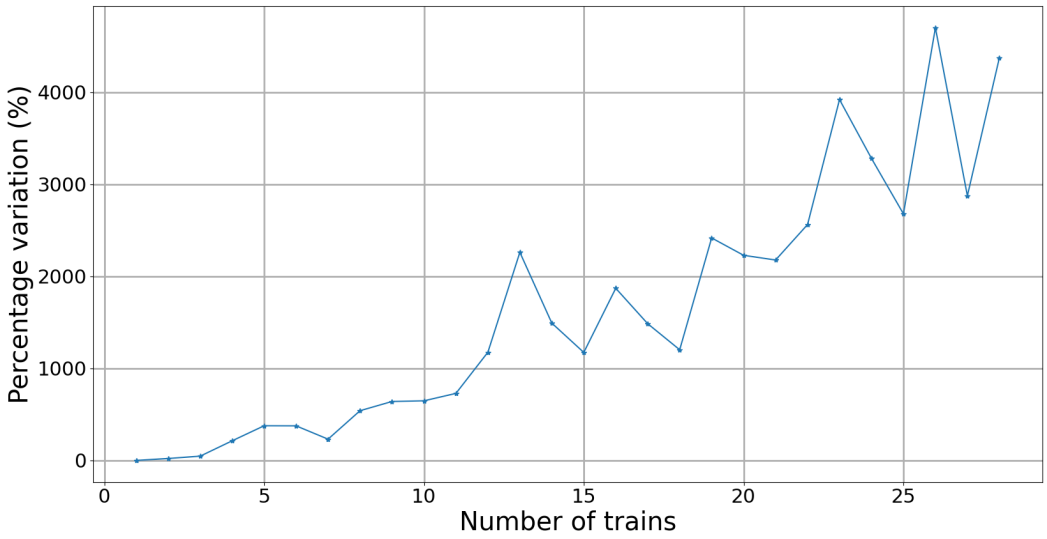


Figure 6.5: A line chart showing the trend, in the case of a track circuit unavailability, of the percentage variation between the CPU-planning time of the replanning strategy and the CPU-planning time of the plan repair strategy when the number of trains in a scenario changes.

In both charts, the values on the y-axis represent the average of the percentage variation in CPU-planning time for scenarios with the same number of trains. On the x-axis, the number of trains per scenario is represented.

The two charts show a similar trend, which can be explained by the similarity of the encoding of the two different case of disruption. Both show an increasing trend: when the number of trains in a scenario increases, the variation in percentage of CPU-planning time between plan repair and replanning increases linearly. For example, on average, for a scenario with 15 trains, the rescheduling operation takes 30 times more CPU-planning time than the plan repair operation.

Another statistic considered in these two analyses concerns the metric quality variation of the plans found between the two strategies.

Figures 6.7 and 6.8 show two histograms with the distribution of the percentage variation between the metrics found in the same scenario between a plan found with a replanning strategy and a plan found with the plan repair strategy without any kind of optimisation process. The percentage variations of metrics have been grouped into 50 uniform bins. In both graphs, we have on the y-axis the number of scenarios for a same bin. On the x-axis, we have the percentage variation value.

Again, we have similar distributions between the two analyses. In particular, it is clear that the highest distribution is around a percentage variation of 0%. However, there are cases with a

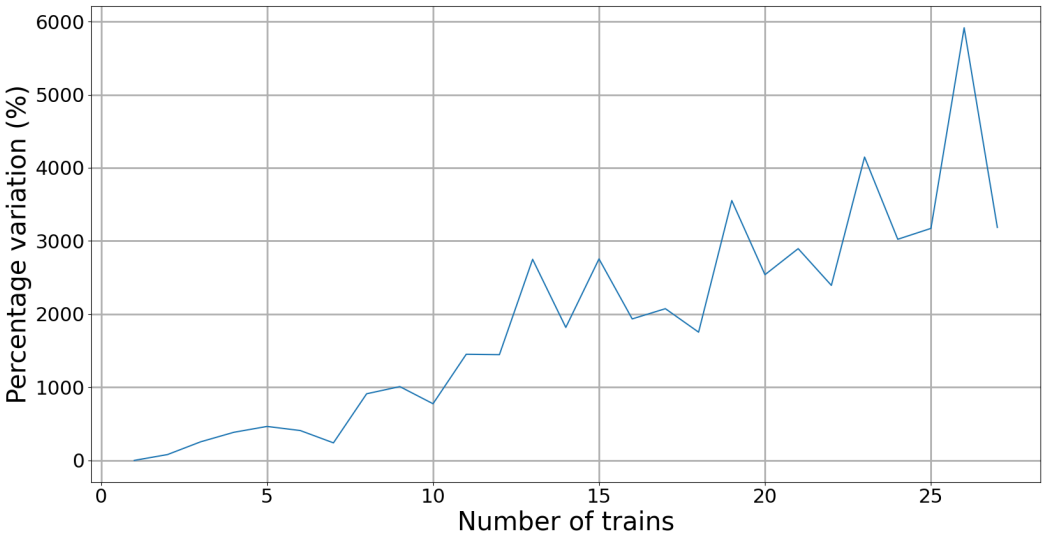


Figure 6.6: A line chart showing the trend, in the case of platform unavailability, of the percentage variation between the CPU-planning time of the replanning strategy and the CPU-planning time of the plan repair strategy when the number of trains in a scenario changes.

percentage difference between a range of -10% and +10%. This implies that, on average, the use of the plan repair strategy does not produce changes in plan quality compared to the use of the replanning strategy.

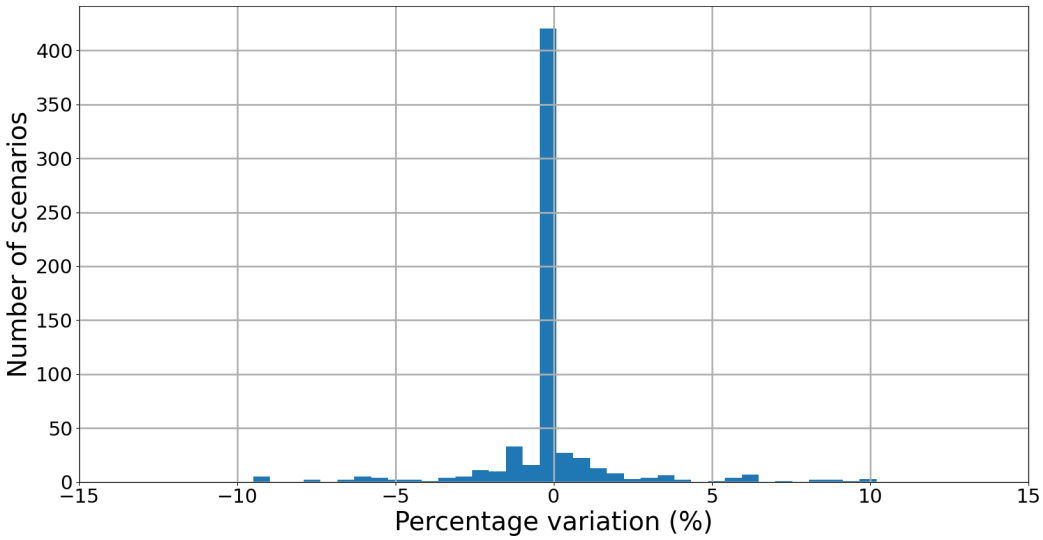


Figure 6.7: The histogram in the figure shows the distribution of the percentage variation between the final plan metric found by the replanning strategy and the final metric found by the plan repair strategy calculated on a series of disruption scenarios with the unavailability of a track circuit.

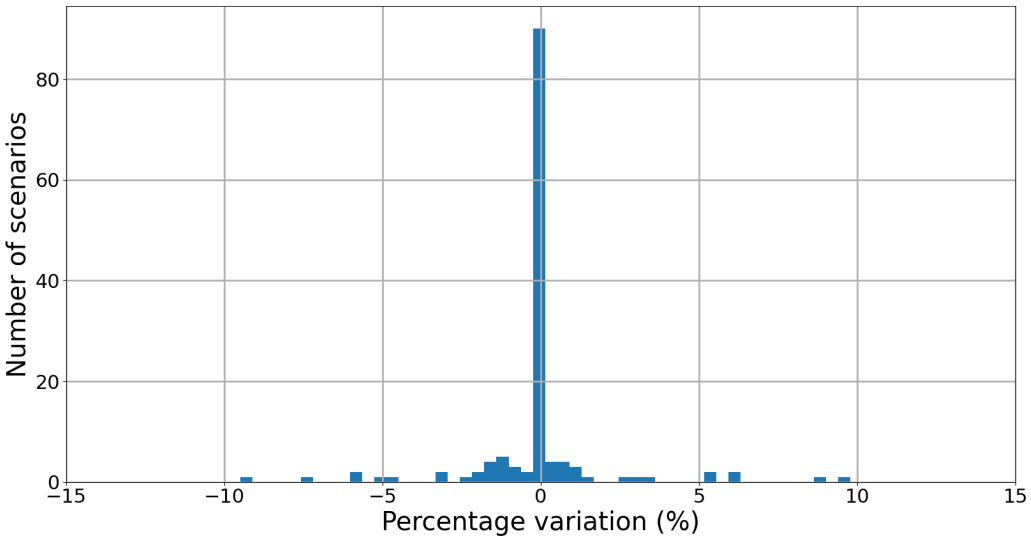


Figure 6.8: The histogram in the figure shows the distribution of the percentage variation between the final plan metric found by the replanning strategy and the final metric found by the plan repair strategy, calculated on a series of disruption scenarios with the unavailability of a platform.

With regard to the analysis concerning the addition of a train not included in the timetable, different days have been considered and for each of them time windows have been extracted in order to define instances, or scenarios, with a number of trains spanning from 1 to 30. For each scenario, one third sub-scenarios have been created in which for each one the original scenario has been copied by removing a random train. The process has been implemented so that each sub-scenario removes a different train than the others. For each scenario and sub-scenario, `PDDL+` files (domain file and problem file) and a heuristics file have been created.

- For each original scenario, a plan is searched through the replanning strategy, so as if it were a standard dispatching problem.
- Whereas, for each sub-scenario, a valid plan has been searched and on the basis of the resulting plan, the operation of transforming actions into events has been applied by re-adding the previously removed train as described in Section 6.2. A valid plan is then searched for using the new scenarios constructed through the plan repair strategy.

The operation of removing one train each time for one third of the trains in the scenario is justified since the complexity at the planning level could differ depending on where a train is inserted. For example, a train that is inserted at the beginning of the scenario schedule may

complicate the planning more than an insertion at the end of the schedule. Hence, the idea is to randomise the placement of the train and to consider a final average value. This procedure simulates the addition of a train to a plan already defined (plan repair strategy) and so, it is possible to compares it with a plan found in the classical way (replanning), represented by the plan found from the original initial scenario.

Similarly to the previous analyses, a comparison between the two strategies is made for this disruption scenario, taking into account the variation in CPU-planning time and differences in plan quality between the plans found.

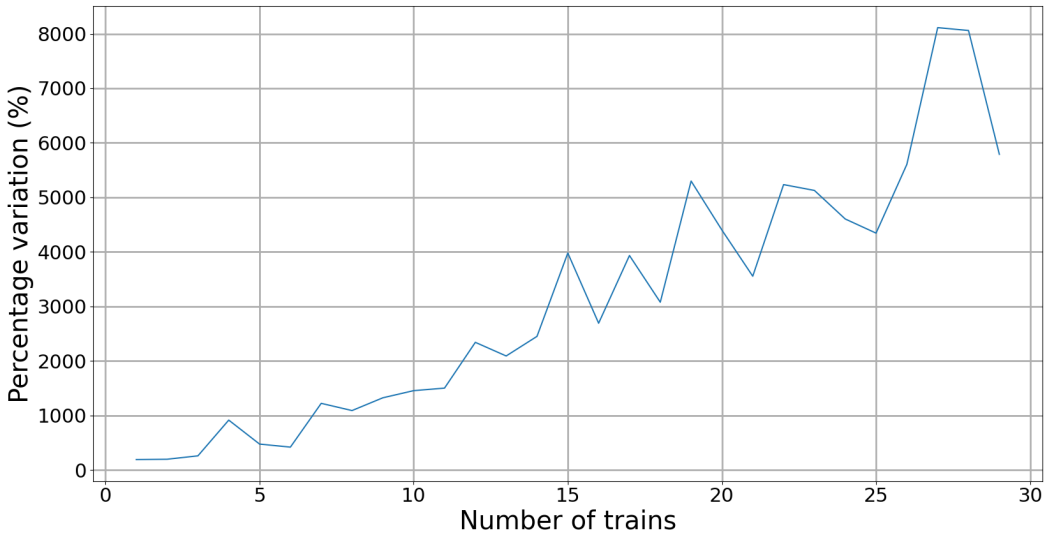


Figure 6.9: The trend, in the case of new train in scenario schedule, of the percentage variation between the CPU-planning time of the replanning strategy and the CPU-planning time of the plan repair strategy when the number of trains in a scenario changes.

Again, the chart in Figure 6.9 shows a linear increasing trend in the percentage variation between the CPU-planning time of the plan repair and the CPU-planning time of the replanning when the number of trains increases. The histogram in Figure 6.10 shows that the metric variation of the plan repair strategy compared to the replanning strategy is almost 0%, i.e. the found plans have the same quality. There are cases in which the quality of the plan obtained from plan repair varies between -10% and +10%. Even in this scenario, the application of the plan repair technique, on average, doesn't produce variation in plan quality compared to the use of the replanning strategy.

Considering all the examined disruption scenarios, it is evident the high efficiency in terms of CPU-planning time of the plan repair strategy with respect to the replanning strategy when the number of trains increases. The motivation is related to the operation of transforming actions into events, which reduces the search space according to the number of fixed actions.

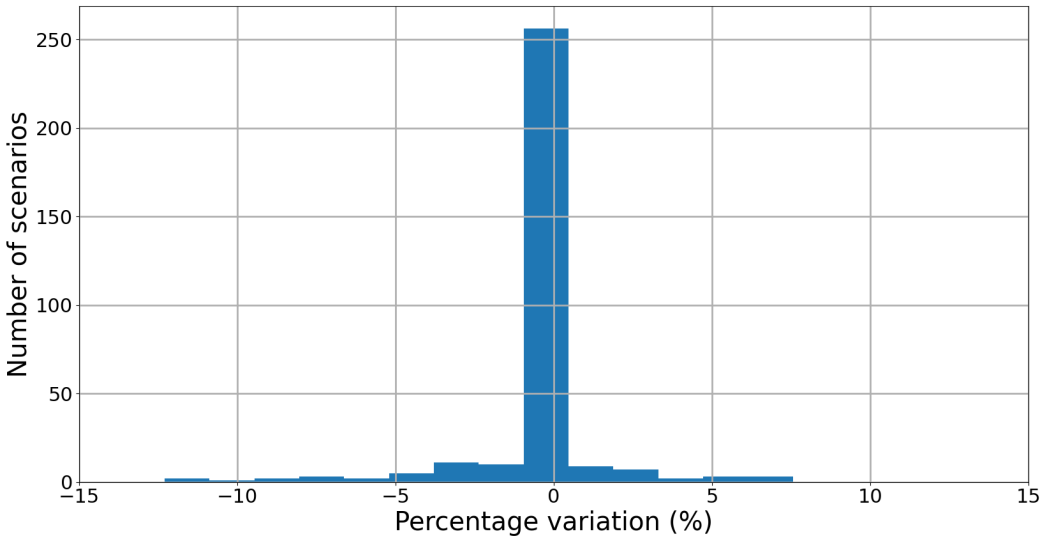


Figure 6.10: The distribution of the percentage variation between the final plan metric found by the replanning strategy and the final metric found by the plan repair strategy, calculated on a series of disruption scenarios with the presence of a new train.

The quality of plans found by the plan repair does not tend to vary with respect to the quality of plans found by the replanning strategy. Variations are symmetrical between -10% and +10%. Thus, in addition to the fact that the CPU-planning time is better, the quality of the plan found by the plan repair strategy on average has no benefit or drawbacks over the quality of the replanning strategy.

The great benefits of this implementation are countered by the fact that, by anchoring train path actions, the search space is reduced. A smaller search space therefore does not guarantee that, once a reference metric is fixed, an optimal plan will be found. As an example, let us consider a rescheduling problem of a scenario with 10 trains and through the operation of transforming actions into events we fix the path of 7 trains. If we apply an optimisation strategy that guarantees the optimum, we are certain to find an optimal plan for the problem created through the action to event operation, but it is not guaranteed to find the optimum plan of the In-Station Train Dispatching problem related to the disruption scenario since the planner does not consider in its search combinations of train paths, involved in the operation of transforming actions into events, other than the ones fixed.

Moreover, in the worst case, it does not guarantee the finding of a valid plan for the scenario with the new conditions. In this case, a possible solution could be a relaxation of the problem with the possibility of not applying the operation of transforming actions into events to the paths of trains temporally close to the trains related to the disruption scenario. In this case, the aim would be to

find a trade-off between the efficiency of the operation and the reduction of its search space.

Chapter 7

Visualiser and other analyses

7.1 Web application: visualiser

A graphical representation of the station can be useful for a better comprehension of its characteristics, an easy visualisation of the allowed train movements and the simulation of specific operations on different scenarios. For these reasons, a web application is implemented using mainly the JavaScript library for data manipulation and visualisation *D3.js*[1].

The web application has two objectives: (i) the possibility of providing an interactive description and identification of the station components, (ii) the implementation of simulators of the adopted strategies for rescheduling and optimisation.

The visualisation tool has been implemented using the *D3.js* library and has the following functionality:

- Possibility of highlighting key components such as track segments, itineraries within the station graphic.
- Possibility to display the result of an In-Station Train Dispatching problem planning. Each train is identified by a different colour, and the chosen itineraries to achieve each goal are displayed over time.

The implemented simulators use the visualisation tool as the output of their results. Three simulators have been implemented to simulate respectively: (i) the unavailability of a track circuit, (ii) the unavailability of a platform, (iii) the optimisation of a plan. It is possible to choose for each simulator various scenarios different from each other.

[1] Data-Driven Documents `https://d3js.org/`

In the first two simulators, it is possible to select either a specific segment track or a specific platform to force their unavailability. The simulator will then show the train movements described in the calculated plan on the basis of their new condition, and it will be possible to see information about the plan as the planning time, the metric and whether a plan is optimal or not on a table in the web page.

Similarly, the optimisation simulator calculates the optimised plan of the chosen scenario, it shows its movements on the representation of the station and the reference metric of the plan, the planning time, and the plan optimality.

At the implementation level, the web application is connected to a server built with the *JavaScript* run-time environment *Node.js*. It processes the request in real time, searches for a valid plan based on the chosen scenario and the imposed condition. The server returns a data structure containing all train movements and a plan data. The web application uses the data structure to represent the movements and display the information data. A diagram of the process just described is shown in the figure 7.1.

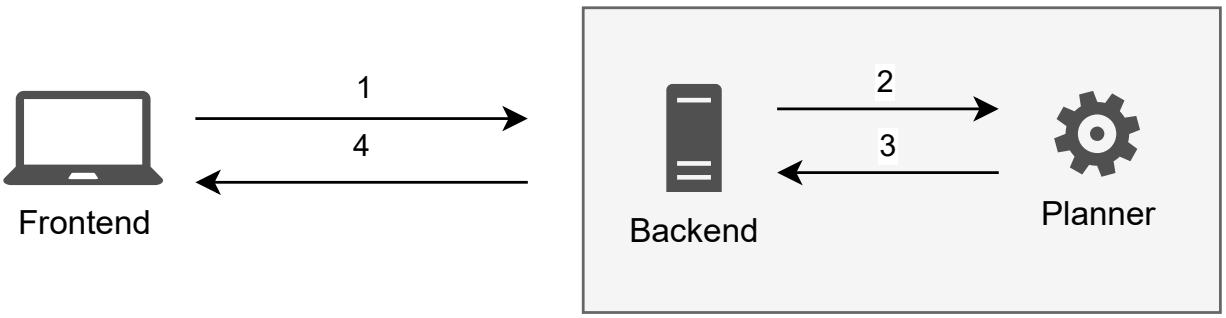


Figure 7.1: Diagram of the information flow for requesting a plan. Once the customised request has been produced, it is handled and processed by the backend. The problem is then sent to the planner which, if it finds a valid plan, sends it as a response to the backend. The backend inserts the plan and other information into a data structure and sends it to the frontend for the visualisation.

The main page of the web application is dedicated to displaying the station components. In particular, an itinerary or a routing can be selected from the lists and displayed on the station representation. By hovering over a track circuit, it is selected with a colour and the name is shown. The same happens if the track circuit identifier is placed inside the search box. Figure 7.2 shows a screenshot of this page of the web application.

The page with the embedded simulator is shown in the Figure 7.3. After selecting a specific scenario, it is possible to: either request a plan with no disruption scenario, or request a plan by selecting either an unavailable track circuit (by clicking on one of the graphic) or an unavailable platform from the list. The optimise button asks for an optimised plan of the selected scenario. The calculated plan is then displayed on the graphic showing over time of the train movements.

Each train is associated with a colour, and each itinerary movement with the associate action or event is shown on the graphic. Through a table it is possible to view some metrics related to the plan such as its planning time, the metric, and whether the plan is optimal or not. In a text box, it is possible to insert the result of a planning to display it.

In the described screenshots, in order to maintain the data confidentiality, the representation of the real station used for analysis has been replaced with an illustration of a non-real station.

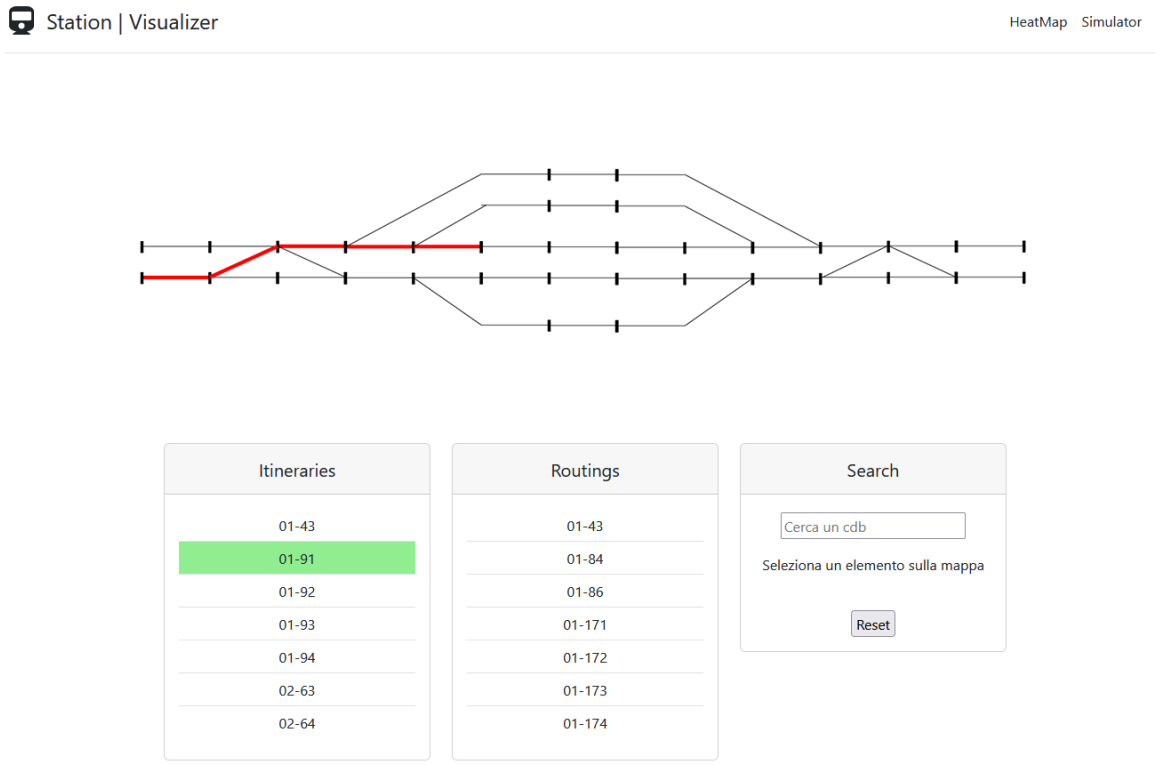


Figure 7.2: The main page of the web application with the visualisation tools for station components.

7.2 Criticality of a segment track

The most important railway service disruptions presented in this analysis is certainly the non-availability of a track circuit. This unavailability, in practice, can be realised, for example, by the physical breaking of the track or the presence of an unwanted object on it or other circumstances. Due to the nature of stations, it is evident that not all track segments have the same importance. Some of them are essential for ordinary activities. For example, the non-availability of a segment

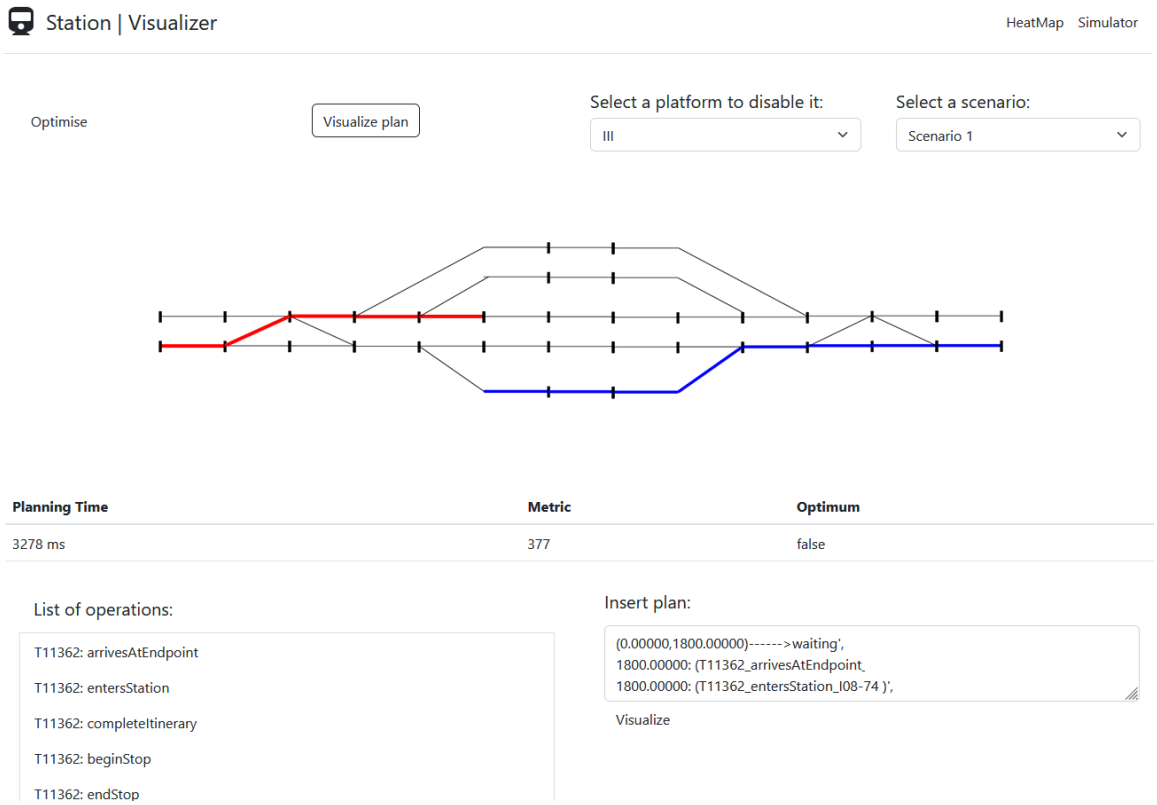


Figure 7.3: The page of the web application with the simulation and visualisation tools for the In-Station Train Dispatching problem.

track placed at the entrance or exit of the station will produce a more critical situation than the non-availability of a segment track in the centre of the station.

In order to prevent, but also to intervene in the most functional way, it is necessary to have knowledge of the criticality due to disruption in the event of a track segment breaking or being occupied.

We define a track segment as essential if, in the case of its unavailability, at least one itinerary containing it could not be replaced with another itinerary. For example, the non-availability of a track segment at the beginning of an entry itinerary is most likely essential.

A first static analysis, i.e. not based on train movement data, can be made by analysing the composition of all itineraries within the station: for each itinerary, the repetitions of each track segment are counted. The idea is that if many itineraries share the same track segment, then blocking it produces a higher criticality.

Another type of analysis, however, can be done through data analysis of train movements in stations found by planning: having identified a set of scenarios, a set of sub scenarios is created

for each track segment. Then, for each sub scenario, only one track segment is blocked. At the end, a plan is searched that can solve the new sub scenario. Once the data is obtained, it is counted, for each track segment, how many times its failure led to a plan not being found. At this point the more times this track segment has produced the non-finding of a plan the more critical the track segment is. This type of analysis provides a classification of the most critical essential track segments, i.e. those that are used most often.

Similarly, it is possible to evaluate the criticality of non-essential track segments. Also in this case, a series of scenarios are considered and a sub-scenario is created for each track segment that causes its non-availability.

A different plan from the original one that does not require the track segment is searched for. For each segment track where it is possible to find a plan to replace it, the percentage difference in CPU-planning time between the planning of the original scenario and the related sub-scenario is calculated. Intuitively, the higher the percentage difference, the more difficult it was to find a new path. An average is then made for all segment track of all scenarios considered, giving a criticality score for all non-essential track circuits in the station.

Chapter 8

Related Work

8.1 Rescheduling problems

The object of rescheduling a dispatching problem is very often associated with real-time timetable rescheduling problems. An approach such as [Binder et al., 2017] proposes an Integer Linear Program (ILP) model to reschedule the timetable considering as objectives the passenger satisfaction, the operational costs and the deviation from the undisrupted timetable. A similar approach of an Integer Linear Program (ILP) model to solve the real-time railway timetable rescheduling problems is described in [Veelenturf et al., 2014]. In this analysis, a large network consisting of stations and their connections is considered. It is analysed only the scenario in which tracks are occupied, and it is proposed, as solution, the possibility of re-routing trains to reduce the delay or the number of cancelled trains. Each train service like departure or arrival is represented by an event, and the main idea is to minimize a function composed by a weighted sum of the number of cancelled trains and the sum of the delays of all the events from their original scheduled times. Others, instead, such as [Rodrigo Acuna-Agost and Guey, 2011] proposes a MIP formulation applicable to the local context of a single station to tackle the problem of finding a new train schedule after a case of disruptions. The particularity is that the proposed approaches use the original, no longer valid, scheduling to create the new one. More specifically, one strategy, right-shift rescheduling, fixes certain integer variables to maintain a certain order of trains. Another strategy is to use the original scheduling to produce local branching cuts to add to the MIP model. [Dollevoet et al., 2017], instead, proposes an iterative framework to find a new feasible solution by considering combinations of rescheduling resources such as timetable, crew or rolling stock after the occurrence of a disruption. Most of the cited works focus on rescheduling analysis through a more extended approach of the railway network. Similarly, [Kecman et al., 2013] attempts to analyse possible solutions in the event of propagating delays or deviations from the original timetable at different network level as, for instance, by extending its working inputs to the entire Dutch national railway network, using alternative graph strategies and so-

phisticated metaheuristics. A particular work has been presented by [Li] that propose MAPF (Multi-Agent Path Finding) algorithms for planning and replanning applied for a Flatland challenge with the representation of large-scale rail networks, but which can nevertheless be applied to more real-world scenarios.

8.2 Optimisation problems

In a dispatching problem, the reference metrics may be different and, for example, relate to train time at the station, as in our analysis, or train delay, component utilisation, etc... Many techniques used to optimise these metrics are not specifically related to the problem domain, but to generic optimisation techniques of a strategy through the encoding of the problem. For example, in [Suhl et al., 2001] is considered minimization of total delay over waiting time for passengers while maintaining the quality of a schedule (frequency of service, good connections) using models based on time-space networks. In [Abels et al., 2021] it is presented a hybrid approach that extends Answer Set Programming (ASP) [Lifschitz, 1999] to tackle large-scale, real-world train scheduling instances involving an optimisation technique, minimizing the delay.

8.3 Rescheduling strategies in other domains

There are different strategies used to reschedule a plan that is no longer valid, and their effectiveness may depend on the domain analysed. [Arangu et al., 2008] describes a repair technique able to identify the failure, remove the actions not more valid from the domain and schedule, allowing to reuse the actions of the current plan. Similarly, [Krogt and Weerdt, 2005] proposes an *unrefinement* operation in which the invalid actions are removed and an operation to add the actions necessary to achieve the goal. The focus of the strategy is the removal of actions, which is achieved through a planning heuristic. It [Boella and Damiano, 2002] also exploits the planner's ability to evaluate the goodness of partial plans so that, if the plan is no longer valid, the original plan can be reduced until a good sub-plan is found from which a valid plan can be reconstructed. Other strategies instead are integrated with the search strategy used. For example, [Koenig et al., 2002] uses an own version of LPA* (lifelong planning A*) which preserves information concerning the construction of the search tree of the original plan. As soon as there are changes in the initial conditions, the heuristics of the original search tree are also updated. A search for a new valid goal state can then be resumed. Another strategy described in [Fox et al., 2006] uses an adapted LPG, a local-based planner [Gerevini et al., 2003], with a modified evaluation function to emphasise the importance of plan stability. The evaluation function penalises changes from the original input plan maintaining the plan quality and temporal compactness.One of the best known approaches to plan repair problems is CHEF, Case-based planner [Hammond, 1986]. The system was designed specifically for the cooking domain. The strategy initially identifies a set

of strategies for solving different problems in the domain. When the system identifies a failure, CHEF creates an explanation of why it happened and searches for a repair strategy based on the cause.

Chapter 9

Conclusions and Future Work

9.1 Conclusion

In this thesis, two extensions of the automatic solution of the In-Station Train Dispatching problem have been presented. The first extension concerns the optimisation operation of a plan and it has been analysed by first defining a metric and then presenting different optimisation implementation strategies. The second extension concerns the operation of rescheduling an invalid plan by comparing two strategies: (i) replanning, or planning from scratch, and (ii) plan repair, i.e. a repair of the invalid plan using information from the original plan. The analysis focused mainly on the latter strategy by presenting an implementation of it, and the action-to-event transformation operation. For both extensions, an analysis has been made on real-world historical data of a medium-sized railway station from North-West of Italy, provided by RFI. In the first case, the significant potential of the dynamics constraints and gradient descent strategies to iteratively optimise a plan metric has been shown. In the second case, also through the definition of three possible disruption scenarios, the enormous effectiveness in terms of reducing CPU-planning time while maintaining a similar quality of plan between plan repair strategy and rescheduling strategy has been shown.

9.2 Future Work

In this thesis several proposed strategies have been presented, highlighting the positive and negative aspects of each. However, the field of study for both topics is very large and offers numerous suggestions for a continuation of the analysis. This section will propose different future works that can be approached as a continuation of this analysis.

9.2.1 Optimisation

Improvement of the iterative process. The analysis related to the optimisation of the In-Station Train Dispatching problem in Chapter 5 is mainly dedicated to the identification of a metric to be optimised and some strategies, not yet advanced, for the process of optimising a plan. All the presented strategies are identified in an iterative process, which searches for a plan each iteration by constraining the metric more and more. This process has the disadvantage of visiting at each iteration nodes already visited in the previous iteration. A possible future study could be related to making the process more efficient by having the plan search information more shared across iterations. An implementation of such a process inside the planner would also avoid the use of a preprocessor that involves additional computational time and the export of search data between two different systems.

Improvement of presented strategies. From the results presented in Section 5.3, the potential of dynamic constraints and gradient descent strategies has been highlighted. Currently, their implementation can certainly be improved to ensure more efficient execution.

With regard to dynamic constraints, these found their efficiency on the degree of prediction of future train movements within the station. A higher prediction capacity, e.g. through logical implications, machine learning, neural networks, can increase the constraint restriction and consequently the efficiency of finding an optimal plan. Furthermore, in the analysis, dynamic constraints are only performed on the total metric that refers to the sum of train times at the station. A possible future analysis may be to use the technique on individual train times.

Regarding the gradient descent strategy, a possible improvement to reduce the planning time could concern the possibility of anchoring the trains paths, through the action-to-events operation, identified for rescheduling operations and described in Section 6.2, of the trains for which we are sure that we already have an optimal path. Since they are not binding on each other, the two strategies can be combined, bringing with them their relative advantages, in terms of planning time, and disadvantages, since the application of the gradient descent strategy does not guarantee the finding of an optimum solution.

9.2.2 Rescheduling

New disruption cases and modelling of more real-world aspects. In this analysis concerning rescheduling, we have focused on its application to three specific disruption scenarios. However, the disruptions or issues that can affect trains inside a train station or, more generally, the rail network are several. Some examples mentioned in [Fang et al., 2015] are, for instance, a change in departure or arrival times, a change related to unscheduled stops, or a change related to travel times on a specific itinerary. For this reason, it is necessary to investigate new disruption scenarios and define a consistent encoding. In addition, the analysis, in order to reduce the complexity

of the problem, has focused on the direct effect of the disruption, not taking into account possible secondary and indirect effects on other elements of the station or relations with other types of disruptions. These aspects can be taken into account in a future analysis to be able to solve increasingly complex scenarios and to produce the most realistic solution possible.

New plan repair implementations and real-time rescheduling . The results presented in Section 6.4, regarding the significant difference in planning time between planning from scratch and plan repair implementation, highlight the great usefulness of adopting a solution such as the latter, especially in very complex scenarios. The important advantage related to the reduction in planning time is contrasted by the reduction in search space and therefore the non-guarantee of finding a valid plan. A possible future study could involve new implementations of plan repair through new information from the initial plan that is no longer valid. Some of them, for example, may involve maintaining the search structure used to find the initial plan in order to, not only try to keep the search space complete, but also to avoid visiting, in the new search, nodes already visited in the initial search.

An increasingly efficient rescheduling technique combined with a correct and accurate timely detection of the disruption scenario could define, in the future, a real time rescheduling system such that an AI planning agent can directly suggest a new valid solution at each change of the external environment reducing the support of the human dispatcher.

Acknowledgements

Eccoci arrivati alla fine di questo intenso lavoro che termina un percorso importante che se tornassi indietro rifarei senza dubbi. Ci tengo allora a ringraziare le persone che in tutto questo tempo hanno contribuito a renderlo speciale e unico.

Un ringraziamento dovuto è al mio relatore, il Prof. Maratea. Dal lato professionale perché è stato un ottimo insegnante capace di farmi appassionare alla materia e capace di seguirmi pazientemente in questo lavoro. Dal lato umano perché ha sempre trovato un momento per ascoltare i miei dubbi e ha sempre valorizzato le mie qualità.

Ringrazio il mio correlatore, Dott. Cardellini. Questo lavoro non sarebbe esistito senza i suoi studi, insegnamenti e consigli. Lo ringrazio soprattutto per la pazienza dimostrata nell'avermi seguito meticolosamente rispondendo sempre in maniera completa alle mie domande.

Ringrazio tutta la mia famiglia senza la quale oggi non sarei qui. In particolare ringrazio mamma e papà che mi hanno cresciuto, educato e permesso, supportandomi, di seguire sempre le mie passioni.

Ringrazio Marco, il fratello che ogni persona dovrebbe avere. Una fonte inesauribile di consigli (non solo di lego) e una persona a cui poter confidare tutto.

Ringrazio tutti i miei nonni perché con le loro parole e i loro gesti mi hanno sempre fatto sentire speciale e accolto.

Penso sia scontato e forse riduttivo un ringraziamento a Irene. Mi ha accompagnato ogni singolo giorno sopportando ogni mio discorso senza fine e ogni mio silenzio, dandomi la forza in ogni momento buio e gioia nei momenti più belli. Devo a lei tanto.

Ringrazio Walter, compagno di viaggio e di avventure. Ho avuto la fortuna di condividere con lui la maggior parte dei momenti della mia vita. Grazie per i discorsi e le risate assieme, gli scleri pre-esami e per essere, oltre a tutto, una persona sempre presente.

Ringrazio gli amici Elisabetta, Erika, Federico, Giada, Giulia, Giovanni, Ludovica, Marta, Nicolò, Simone e Vittoria che in questi anni mi hanno accompagnato, insieme a molte risate, in esperienze intense della mia vita e hanno contribuito a rendere la persona che oggi sono.

Un ringraziamento agli amici di Oldskool per la compagnia e i momenti di leggerezza e divertimento in questi anni.

E per terminare, ci tengo a rivolgere un particolare ringraziamento a tutte le figure che nella mia

vita hanno contribuito alla mia crescita personale e culturale. In particolare, un ringraziamento speciale alla professoressa Massa e la professoressa Monti capaci di farmi prendere coscienza delle mie potenzialità e dell'importanza dello studio e della cultura nella vita.

List of Figures

Bibliography

[Li,] Scalable rail planning and replanning: Winning the 2020 flatland challenge. 31:477–485.

[Abels et al., 2021] Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., and Wanko, P. (2021). Train scheduling with hybrid answer set programming. *Theory and Practice of Logic Programming*, 21(3):317–347.

[Arangu et al., 2008] Arangu, M., Garrido, A., and Onaindia, E. (2008). A general technique for plan repair. volume 1, pages 515–518.

[Binder et al., 2017] Binder, S., Maknoon, Y., and Bierlaire, M. (2017). The multi-objective railway timetable rescheduling problem. *Transportation Research Part C: Emerging Technologies*, 78:78–94.

[Boella and Damiano, 2002] Boella, G. and Damiano, R. (2002). A replanning algorithm for a reactive agent architecture. pages 183–192.

[Cardellini et al., 2021a] Cardellini, M., Maratea, M., Vallati, M., Boleto, G., and Oneto, L. (2021a). An efficient hybrid planning framework for in-station train dispatching. In *Computational Science - ICCS 2021 - 21st International Conference, Krakow, Poland, June 16-18, 2021, Proceedings, Part I*, volume 12742 of *Lecture Notes in Computer Science*, pages 168–182. Springer.

[Cardellini et al., 2021b] Cardellini, M., Maratea, M., Vallati, M., Boleto, G., and Oneto, L. (2021b). In-station train dispatching: A PDDL+ planning approach. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, pages 450–458. AAAI Press.

[Dollevoet et al., 2017] Dollevoet, T., Huisman, D., Kroon, L. G., Veelenturf, L. P., and Wagenaar, J. C. (2017). Application of an iterative framework for real-time railway rescheduling. *Computers Operations Research*, 78:203–217.

[Fang et al., 2015] 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.

[for Mobility and Transport, 2019] for Mobility, D.-G. and Transport (2019). Transport in the european union: Current trends and issues.

[for Railways (EU body or agency), 2022] for Railways (EU body or agency), E. U. A. (2022). *Report on railway safety and interoperability in the EU, 2022*. Publications Office of the European Union.

[Fox et al., 2006] Fox, M., Gerevini, A., Long, D., and Serina, I. (2006). Plan stability: Replanning versus plan repair. volume 2006, pages 212–221.

[Fox et al., 2005] Fox, M., Howey, R., and Long, D. (2005). Validating plans in the context of processes and exogenous events. pages 1151–1156.

[Fox and Long, 2006a] Fox, M. and Long, D. (2006a). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297.

[Fox and Long, 2006b] Fox, M. and Long, D. (2006b). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297.

[Gerevini et al., 2003] Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290.

[Givoni et al., 2009] Givoni, M., Brand, C., and Watkiss, P. (2009). Are railways climate friendly? *Built Environment*, 35(1):70–86.

[Hammond, 1986] Hammond, K. J. (1986). Chef: A model of case-based planning. In *AAAI*.

[Kecman et al., 2013] Kecman, P., Corman, F., D'Ariano, A., and Goverde, R. (2013). Rescheduling models for railway traffic management in large-scale networks. *Public Transport*, 5.

[Koenig et al., 2002] Koenig, S., Furcy, D., and Bauer, C. (2002). Heuristic search-based replanning. AIPS'02, page 294–301. AAAI Press.

[Krogt and Weerdt, 2005] Krogt, R. and Weerdt, M. (2005). Plan repair as an extension of planning. pages 161–170.

[Laiton-Bonadiez et al., 2022] Laiton-Bonadiez, C., Branch-Bedoya, J. W., Zapata-Cortes, J., Paipa-Sanabria, E., and Arango-Serna, M. (2022). Industry 4.0 technologies applied to the rail transportation industry: A systematic review. *Sensors*, 22(7).

[Lifschitz, 1999] Lifschitz, V. (1999). Answer set planning. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 373–374. Springer.

[Mcdermott et al., 1998] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control.

[Moscarelli et al., 2017] Moscarelli, R., Pileri, P., and Giacomel, A. (2017). Regenerating small and medium sized stations in italian inland areas by the opportunity of the cycle tourism, as territorial infrastructure. *City, Territory and Architecture*, 4(1):1–14.

[Rodrigo Acuna-Agost and Guey, 2011] Rodrigo Acuna-Agost, Philippe Michelon, D. F. and Guey, S. (2011). A mip-based local search method for the railway rescheduling problem. *Wiley Periodicals*, 57(1):69—-86.

[Scala et al., 2016] Scala, E., Haslum, P., Thiébaux, S., and Ramirez, M. (2016). Interval-based relaxation for general numeric planning. In *Proceedings of the European Conference on Artificial Intelligence*, pages 655–663.

[Scala et al., 2020] Scala, E., Haslum, P., Thiébaux, S., and Ramírez, M. (2020). Subgoaling techniques for satisficing and optimal numeric planning. *Journal of Artificial Intelligence Research*, 68:691–752.

[Scalise, 2014] Scalise, J. (2014). How track circuits detect and protect trains. In *Railw. Walk Rail Talk*, volume 1, pages 1–7.

[Suhl et al., 2001] Suhl, L., Mellouli, T., Biederbick, C., and Goecke, J. (2001). *Managing and Preventing Delays in Railway Traffic by Simulation and Optimization*, pages 3–16. Springer US, Boston, MA.

[Veelenturf et al., 2014] Veelenturf, L., Kidd, M., Cacchiani, V., Kroon, L., and Toth, P. (2014). A railway timetable rescheduling approach for handling large scale disruptions. *SSRN Electronic Journal*.