In-Station Train Dispatching: A PDDL+ Planning Approach

Matteo Cardellini,[1] Marco Maratea,[1] Mauro Vallati,[2] Gianluca Boleto,[1] Luca Oneto[1]
[1] DIBRIS, University of Genoa, Italy
[2] School of Computing and Engineering, University of Huddersfield, UK
matteo.cardellini@edu.unige.it, marco.maratea@unige.it, m.vallati@hud.ac.uk,
gianluca.boleto@edu.unige.it, luca.oneto@unige.it

Abstract

In railway networks, stations are probably the most critical points for interconnecting trains' routes: in a restricted geographical area, a potentially large number of trains have to stop according to an official timetable, with the concrete risk of accumulating delays that can then have a knockout effect on the rest of the network. In this context, in-station train dispatching plays a central role in maximising the effective utilisation of available railway infrastructures and in mitigating the impact of incidents and delays. Unfortunately, in-station train dispatching is still largely handled manually by human operators in charge of a group of stations.

In this paper we make a step towards supporting the operator with some automatic tool, by describing an approach for performing in-station dispatching by means of automated planning techniques. Given the mixed discrete-continuous nature of the problem, we employ PDDL+ for the specification of the problem, and the ENHSP planning engine enhanced by domain-specific solving techniques. Results on a range of scenarios, using real-data of a station of the North West of Italy, show the potential of our approach.

Introduction

Railways play a significant economical role in our society for transporting either goods or passengers, but the increasing volume of people and freight transported on railways is congesting the networks (Bryan, Weisbrod, and Martland 2007). Within a railway network, stations are probably the most critical points for interconnecting trains' paths. This is because, in a station, a number of trains need to stop according to a given timetable, and a number of possible routes are thus occupied, with the concrete risk of accumulating delays, that may boil down to cost penalties and inconveniences for passengers. In the context of railway stations, one of the main problems is how to mitigate delays and deal with incidents with the aim of minimising their overall negative impacts. This problem is generally defined as in-station train dispatching and plays a pivotal role to maximise the capacity of the railway infrastructures. Unfortunately, still today, this problem is handled manually by experienced operators in charge of a large set of connected stations. These operators are in charge of monitoring the conditions, and to give instructions to train conductors with regards to the path to follow, and the platform to reach (if needed). This job is currently receiving very limited support by the railway control systems which provide an abstract overview of the traffic conditions of the station focusing mainly on the safety of the passengers.

In this paper we concentrate on the in-station train dispatching problem and make a significant step towards supporting the operator with a tool able to solve the problem in an automated way by means of automated planning. Given the mixed discrete-continuous nature of the problem, we employ PDDL+ (Fox and Long 2006) for the specification of the models. Notably, PDDL+ has been already successfully exploited in a number of application domains including UAV manoeuvring (Ramírez et al. 2018), battery management (Fox, Long, and Magazzeni 2012), and urban traffic control (McCluskey and Vallati 2017). Given the complexity and the characteristics of the railway application domain, we extended the state-of-the-art planning engine ENHSP (Scala et al. 2016, 2020) with customised techniques such as an adaptive notion of delta, a domain-specific heuristic, and a set of ad-hoc constraints.

For evaluating our solution, we considered real-world historical data of a medium-sized railway station from North-West of Italy provided by Rete Ferroviaria Italiana (RFI),[1] and tested our approach in a large number of scenarios. Results show the potentiality of our solution in each of the evaluated scenario; in particular, our solution (i) models historical data in an accurate way, (ii) is in general able to (partially) absorb delays that trains may have when entering the station, and (iii) can support in evaluating an increment of the railway station capacity. Further, our analysis shows that the proposed extended ENHSP outperforms significantly the basic ENHSP on the instances of this domain.

Research Context

The broader topic of train traffic control in railway networks deals with the problem of finding appropriate routes for trains in order to respect a given official timetable. The traffic control topic can be divided into two main areas: *line*

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]Because of confidentiality issues we cannot report the name of the station and other details regarding the data that cannot be disclosed by RFI.

dispatching and the *in-station train dispatching*. The former considers the overall railway network and focuses on the routing between different railway stations, while the latter is focused on the routing of trains inside a specific station. An orthogonal problem is the generation of the train timetables which involves knowledge at both railway network and station levels.

With respect to line dispatching, Lee and Chen (2009) introduced an heuristic-based approach for tackling the problem of finding routes for train while generating an overall timetable. Böcker, Lind, and Zirkler (2001) explored the use of a multi-agent scheduling system considering the railway transport system as a case study. On the topic of multi-agent, more recently Atzmon, Diei, and Rave (2019) exploited multi-agent path finding to search for a route for a set of trains from a given origin point to a required individual destination. A different line of work exploits problem decomposition, where a master-slave algorithm is used to control the train traffic of large railway networks (Lamorgese and Mannino 2013, 2015; Lamorgese, Mannino, and Piacentini 2016).

For what concerns instead in-station train dispatching, Mannino and Mascis (2009) introduced a mixed-integer linear programming (MILP) model for controlling a metro station. Their experimental analysis demonstrated the ability of the proposed technique to effectively control a metro station but also highlighted scalability issues when it comes to control the much larger and more complex railway stations. More recently, Kumar et al. (2018) introduced a constraint programming model for performing in-station train dispatching in a large Indian terminal: this approach demonstrates to be able to deal with a large railway station, at the cost of considering very short time horizons (less than 10 minutes) and station-specific optimisations.

Given the complexity of the train dispatching problem, many works focused on related sub-problems or on a more abstract formulation of the overall problem. For example, Rodriguez (2007) formulated a constraint programming model for performing train scheduling at a junction, which shares some of the characteristics of a station, but does not include platforms and stops. Differently from Rodriguez (2007), a number of works (Cardillo and Mione 1998; Billionnet 2003; Chakroborty and Vikram 2008) focused on the problem of assigning trains to available platforms, given the timetable and a set of operational constraints. Taking another perspective, Caprara et al. (2010) focused on the identification and evaluation of recovery strategies in case of delays. These strategies include actions such as the use of different platforms or alternative paths.

Finally, the problem of generating train timetables can be further divided into two different levels. The first one, the planning level, consists in generating the railway network timetable over a long period of time (in general seasonal); the resulting timetable is then visible to the passengers. The second one, the operational level, focus on timetable rescheduling which is the daily task of the operators of adjusting the timetable in case of disruptions, maintenance, or addition of trains (usually freight trains). Problems at the first level can be naturally expressed as constraint satisfaction problems. The vast majority of works on this topic exploits MILP models, while considering different set of constraints and different levels of detail (Caprara, Fischetti, and Toth 2002; Barrena et al. 2014; Cacchiani, Furini, and Kidd 2016). A number of domain-specific approaches have also been introduced, and the interested reader is referred to Cacchiani and Toth (2012) for an extensive review of the field. The literature on the second level is much more variegated but less extensive, given the multifaceted nature of the problem to be addressed. D'ariano, Pacciarelli, and Pranzo (2007) introduced a branch and bound algorithm to recompute a conflict-free and feasible timetable, given the current network conditions. A method to avoid conflicts and to early identify unfeasible timetable schedules is presented by D'Ariano, Pranzo, and Hansen (2007). Finally, given a timetable and a current set of delayed trains, Corman et al. (2012) focused on algorithms to explore the trade-off between cancelling trains to reduce congestion and delays, and the inconvenience caused to passengers due to missed connections and limited service.

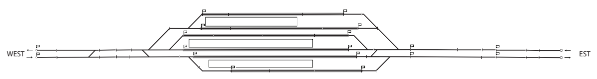

Figure 1: A subsection of the real-world railway station considered in the experimental analysis. Flags are used to indicate itineraries; shorter segments represent the track segments; bold has been used to indicate platform stops.

The In-Station Train Dispatching Problem

We now describe the in-station train dispatching problem, following the formalisation introduced by Lamorgese and Mannino (2015).

A railway *station* can be represented as a graph, composed by a set of connected *track segments*, the minimal controllable rail units. Their status can be checked via track circuits, that provide information about occupation of the segment and about corresponding timings. Track segments can be divided into two classes, stopping points and interlocking. A stopping point is a track segment in which a train can stop: this is if there is a connected *platform*, or at the entrance and the exit point of the railway station. Entry points are segments where the train stops before being allowed into the station (or queues behind other trains); similarly, exit points allow the train to leave the station and enter the outside railway network. Interlocking track segments are segments where a train is not allowed to stop. Sequences of connected track segments are organised in *itineraries*; this is manually done by experts of the specific railway station. Figure 1 provides a schematic representation of part of the Italian railway station we use in our experimental analysis. In the figure, track segments and platforms are easily recognisable, and flags are used to indicate initial and ending points of itineraries.

A track segment can be occupied by a single train at the time. For safety reasons, a train is required to *reserve* an itinerary, and this can be done only if the itinerary is currently not being used by another train. While a train is navigating the itinerary, the track segments left by the train are released. This is done to allow trains to early reserve itineraries even if they share a subset of the track segments.

A train going through the controlled railway station is running a route in the station graph, by reserving an itinerary and moving through the corresponding track segments. In the common case of passenger trains, the route contains one or more track segments connected with a platform. Also, if the train enters (exits) the station, the path will contain an entry (exit) point. Cases where either exit or entrance are missing from a train route include *Origin* trains, that are those whose route starts from the controlled railway station and *Destination* trains, that terminate at the controlled station.

A *timetable* is the schedule that includes information about when trains arrive at the controlled station, when they arrive at a platform, and the time when they leave a platform.

We are now in the position to define the in-station train dispatching problem as follows: Given a railway station infrastructure, a set of trains and their current position within the station or their time of arrival at the controlled station, find a route for every train that allows to respect the provided timetable, as much as possible.

The PDDL+ Domain Model

In this section, we introduce and specify the PDDL+ model we designed for dealing with the problem described in the previous section.

Figure 2 provides a graphical representation of how actions and events are interleaved with regards to different potential states of a train: Parallelograms indicate train states, grey rectangles indicate operators, and white rectangles indicate events. Train states are not explicitly represented in the PDDL+ model, but they can help understanding the structure and the dynamics of the encoding. In particular, a train can be: *Approaching*, ready to enter the controlled station; *Origin*, train that originates from the controlled station – so the train is stopped at a platform; *Destination*, the train terminates at the station; *Intermediate stop*, the train is currently stopped at a platform, and the station is one of its calling points, and finally *Left* indicates that the train has left the controlled railway station.

The core of the proposed PDDL+ encoding is the way in which the movement of trains in the railway station is modelled and controlled. The operators *Enter Station*, *Leave Origin*, and *Begin Overlap* are all allowing the corresponding train to reserve an itinerary and to start moving on it. They have differences in terms of preconditions and effects, as they deal with trains in different logical states. For instance, the *Enter Station* operator can be used only by a train that is at an entry point, one of the effects is that the train is no longer approaching the train station, but is navigating through the controlled station. The use of these operators trigger a process that is used to model the time needed by the train to reach the end of the itinerary. Over time, track segments of the itinerary that are not occupied by the train anymore, are released via the *Completed Track Segment* events.

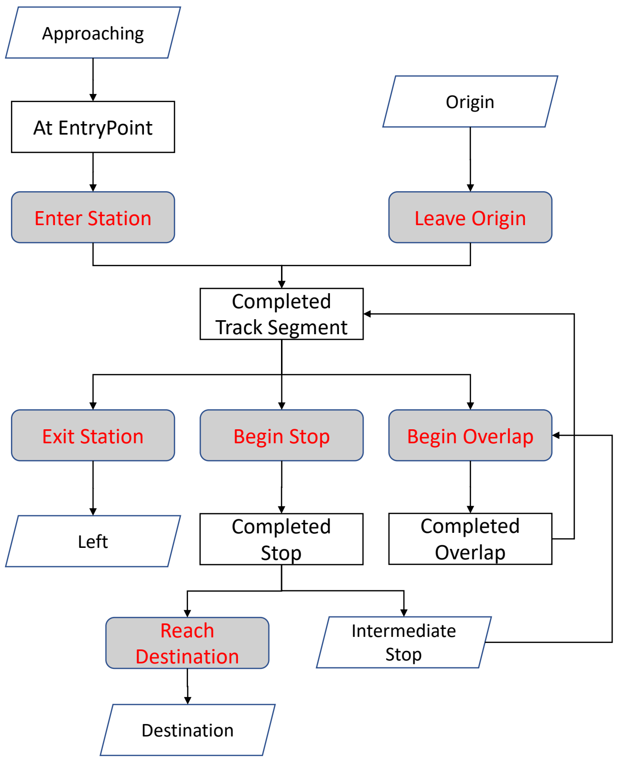

Figure 2: Valid state transitions for a train in a controlled railway station. Parallelograms indicate train states, grey rectangles indicate operators (whose name is in red for readability), and white rectangles indicate events. Edges indicate valid sequences. Processes are omitted for readability.

Notably, when a train reaches the end of an itinerary, it is still occupying part of its segment tracks: the precise number depends on the type of train, and on the length of the segments. The *Begin Overlap* operator is used by a train moving between two subsequent itineraries of the station, to take into account the “overlapping” time needed by a train to completely leave the previous itinerary.

Figure 2 also shows the *Exit Station* operator, that is used to allow a train that reached an exit point to actually leave the station and release the occupied track segments, and the *Begin Stop* operator, that is used to stop a train at a platform to allow the disembarking/embarking of passengers. The duration of the stop is variable: each type of train has a minimum time that is required to stop to safely allow the movement of passengers; further, a train is not allowed to leave a platform before its timetabled leaving time. This is encoded in the corresponding action using appropriate preconditions. Finally, the *Reach Destination* operator is used to model the fact that a destination train has terminated, and should not move any further.

Dedicated processes, not shown in Figure 2 for the sake of readability, are used to keep track of the time spent by a train: (i) navigating an itinerary; (ii) overlapping, i.e., mov-

```
(:action T1_entersStation_I1-2
:parameters()
:precondition (and
 (trainIsAtEndpoint T1 EP1)
 ...
 (not (trainHasExitedStation T1))
 (not (trainHasEnteredStation T1))
 (not (trackSegBlocked cdb1))
 (not (trackSegBlocked dev1))
 ...
 (not (trackSegBlocked cdb6)))
:effect (and
 (not (trainIsAtEndpoint T1 EP1))
 (itineraryIsReserved I1-2)
 (trainInItinerary T1 I1-2)
 (trainHasEnteredStation T1)
 (trackSegBlocked cdb1)
 ...
 (trackSegBlocked cdb6)))


(:event T1_completeTrackSegment_I1-2_cdb1
:parameters()
:precondition (and
 (>= (timeReservedIt I1-2) 29)
 (trainInItinerary T1 I1-2)
 (trackSegBlocked cdb1))
:effect (and
 (not (trackSegBlocked cdb1))))
```

Figure 3: Part of the PDDL+ encoding of the Train Dispatching domain.

ing from one itinerary to the next; and (iii) stopping at a platform. Further, in order to encode an explicit notion of the time that is passing, in our model we employ a dedicated *timePassing* process. This helps when dealing with the time-related aspects of the problems to be solved, by avoiding the need to use timed initial literals.

A pivotal point of the PDDL+ model relates to the way in which the structure of the railway station is encoded. To deal with the potentially huge size of the hybrid problem ground in a non-effective way (Scala and Vallati 2020), due to the presence of itineraries, trains, track circuits, etc. the domain model is fully ground for a given problem to solve. The grounding is done via a pre-processing step that takes into account the actual structure of the network, and the considered trains. The structure of the railway station is directly encoded in the ground actions that are provided in the domain model. As an example, Figure 3 shows a grounded instance of the *Enter Station* operator and of the *Completed Track Segment* event. The action incorporates the structure of the network: it allows to reserve an itinerary connected to the entry point, and to block all the corresponding track segments (this is done via the *trackSegBlocked* predicate list). The ordering of release of track segments is encoded via ground events, such as the one presented in the figure, that are timed accordingly.

A planning problem is specified in terms of the initial location of trains, and of the required goal state. Figure 4 provides and excerpt of the PDDL+ description of the initial and goal states for two hypothetical trains; `T1` that is yet to arrive at the controlled station, and has to do an intermediate stop before leaving, and `T2` that originates from the controlled station. In the case of `T1`, the entry point is described (`EP1`), and the goal indicates that the train has to enter the station (*trainHasEntered*), to do a stop at the station (*trainHasStopped*) and has to leave the station via a specified exit point (*trainExitsStationAt*). The time in which `T1` arrives at the station is not specified in the problem model, as it is encoded via a grounded event in the domain model. After the specified amount of time, the *At EntryPoint* event corresponding to `T1` is triggered, and the train can then be controlled by the planning engine. As soon as the train enters the station, the corresponding action can be executed by the planning engine – the action for train `T1` is shown in Figure 3. Beside other functions used to measure the time spent by the train navigating the station or stopping at platforms, there is also a *trainStayInStation* function, that is used to monitor the overall time spent by the train in the station, and to force the planning engine to optimise routes and stops. This has been done for avoiding undesired behaviours such as trains waiting at entry or exit points – which creates congestion and can have a knockout effect on the rest of the railway network. While the way in which goals are encoded for each train can be seen as redundant, at least in terms of PDDL+, this has been done for supporting the heuristic search, as described in the corresponding section.

```
(:init
...
(trainEntersStationAt T1 EP1)
(= (trainStayInStation T1) 0)
...
(trainIsStopping T2)
(trainIsStoppingAtStop T2 SI)
(stopIsOccupied SI)
(trackSegBlocked cdb12)
(trackSegBlocked cdb13)
(= (trainStayInStation T2) 0)
...

(:goal (and
(trainHasEnteredStation T1)
(trainHasStopped T1)
(<= (trainStayInStation T1) 500)
(trainExitsStationAt T1 Ex01)
...
(<= (trainStayInStation T2) 340)
(trainExitsStationAt T2 Ex03)
...
```

Figure 4: Excerpt of the initial state and goal specification, for a train (`T1`) doing an intermediate stop at, and for a train (`T2`) that originates from, the controlled station.

Figure 4 also shows the initial state specification of train `T2`: this train originates from the controlled station, and is initially stopped at platform `SI`. The platform is therefore occupied (*stopIsOccupied*), as well as the corresponding track segments (*trackSegBlocked*) `cdb12` and `cdb13`.

For this train, the only goal is to leave the station within a given time limit.[2]

Forward Heuristic Search for Train Dispatching

The ENHSP planning engine (Scala et al. 2016, 2020) has been used to solve in-station train dispatching problems encoded in PDDL+. ENHSP is a modular planning engine, and includes a range of off-the-shelf search and heuristic techniques. Overall, ENHSP is a forward search planning engine that deals with continuous processes using the Discretise and Validate approach, where the continuous model is initially discretised, then solved, and finally the found solution is validated against the original continuous model. Discretisation is done on the basis of a given delta, that controls the execution, planning, and validation processes of the planning engine. The delta for execution and planning is used to define, respectively, how often the planning engine is updating the state of the world checking for events and processes, and how often the planning engine is allowed to take a planning decision. The off-the-shelf version of ENHSP proved to be capable of solving prototypical instances, hence demonstrating the feasibility of the approach. To allow ENHSP to solve large and complex PDDL+ in-station train dispatching problems, we then leveraged on its modularity and specialised its behaviour in three ways: we designed an adaptive notion of execution and planning delta, we introduced a domain-specific heuristic, and we added three constraints to prune unpromising areas of the search space.

Adaptive Delta

In the proposed PDDL+ model, it is possible to know a priori when events will be triggered. For *Approaching* trains, the moment in which the *At EntryPoint* event will be triggered is given in the ground formulation. All the other events are the result of actions executed by the planning engine, and the moment in which they will be triggered is only related to the time in which the corresponding action has been executed. This implies that there is no need to use a fixed execution and planning delta, but the delta value can be adjusted according to when the next event will be triggered, or an action will be available. From a planning perspective, it is useless to consider all the steps in between, because the state of the world does not change. This is similar in principle to the approach exploited by decision-epoch planning engines for dealing with temporal planning problems (Cushing, Kambhampati, and Weld 2007).

We designed an *Adaptive Delta Queue*, that is initialised as shown in Algorithm 1. Taking into account the set of ground events E, ground actions A, and the initial state description I, it is possible to identify when to stop the execution to take decisions and update the state of the world. Given the proposed PDDL+ model, in the function GENERATE INITIAL QUEUE it is straightforward to extract the time at which an *At EntryPoint* event will be triggered, and the earliest time at which an action *Leave Origin* will be available. Similarly, it is straightforward to check the presence of trains already in the controlled station, and identify the time of the corresponding events, if any. When an action is executed, a new queue Q is created, and attached to the resulting search state. The function EXTEND QUEUE, shown in Algorithm 1, is used to extend the delta queue of the parent state Q_p taking into account the events TE that will be triggered by the applied action a, and their trigger time.

[2]Anonymised example domain and problem models can be found at https://github.com/matteocarde/icaps2021

Algorithm 1 Adaptive Delta Queue

```
1:  function GENERATE INITIAL QUEUE($E, A, I$)
2:      $Q = $ INITQUEUE()
3:      for $e$ in $E \mid e =$ "$AtEntryPoint$" do
4:          ENQUEUE(Q, $e.time$)
5:      for $a$ in $A \mid e =$ "$LeaveOrigin$" do
6:          ENQUEUE(Q, $a.earlyTime$)
7:      for $p$ in GETACTIVEPROCESSES($I$) do
8:          for $e$ in IDENTIFYEVENTS($p$) do
9:              ENQUEUE($e.time$)
10:     return SORT($Q$)
11:
12: function EXTEND QUEUE($Q_p, a$)
13:     $Q = Q_p$
14:     $TE = $ CALCTRIGEVENTS($a$)
15:     for $e$ in $TE$ do
16:         ENQUEUE($e.time$)
17:     return SORT($Q$)
```

In each state, the next delta step is identified by considering the corresponding delta queue, and by picking up the time of the next event that will be triggered. Between two events, nothing will happen, so there is no need to generate and assess additional states.

Specialised Heuristic

Following the traditional A* settings, the cost of a search state s is calculated as $f(s) = g(s) + h(s)$, where $g(s)$ represents the cost to reach s, while $h(s)$ provides an heuristic estimation of the cost needed to reach a goal state from s. In our specialisation, $g(s)$ is calculated as the elapsed modelled time from the initial state to s. The $h(s)$ is a heuristic calculated according to Equation 1:

$$h(s) = \sum_{t \in N} (\mathsf{reachTime}(t) + \mathsf{penalisation(t)}) \quad (1)$$

where N is the set of trains of the given problem that did not yet achieve their goals at s. The reachTime method measures the time that, starting from the current position and using a Manhattan heuristic, the considered train needs to reach its final destination. The reachTime method returns 0 for trains that have not yet left their origin platform (origin trains) or not yet entered the controlled station (any other train). The penalisation method gives a very high penalisation value P for each goal specified for the considered train t that has not yet been satisfied at state s. For instance, considering the example provided in Figure 4, the train `T1` has initially a penalisation of $3 \times P$, because there are three goals related to the train that are not satisfied in the initial state.

In cases where several different states have the same heuristic value, our approach prioritises the application of the actions *Leave Origin*, *Reach Destination* and *Exit Station*, and penalises the use of the action *Begin Stop* for trains that are not required to stop (i.e., origin trains, trains that already made a stop, trains that are passing through the station but does not have to stop).

Constraints

Finally, the implemented constraints focus on the time spent by the trains. In particular, for every train *t* of a given problem to solve, the following constraints are enforced.

$$\mathsf{stayInStationTime}(t) < MaxStayInStation \tag{2}$$

$$\mathsf{fromArrivalTime}(t) < MaxFromArrival \tag{3}$$

$$\mathsf{stoppingTime}(t) < MaxStoppingTime \tag{4}$$

Equation 2 indicates that a train is not allowed to stay in the station more than a given maximum value. Similarly, Equations 3 and 4 constraint, respectively, the time passed from the arrival of the train in the station, and the time spent stopping at a platform. The idea behind such constraints is to avoid situations where trains are left waiting for long periods of time, occupying valuable resources. The maximum times are calculated a priori, according to historical data, and depends on the structure of the railway station. Such constraints are also used to implement an anytime planning framework, able to generate solution of increasing quality over time. Starting from an initial value corresponding to worst case scenarios observed in historical data, the constraints' value are then reduced if a plan is generated: this process is repeated until either the cutoff time is reached, or the planning engine returns that no solution can be found.

Evaluation

We tested the proposed PDDL+-based approach by exploiting the real-world data coming from the Italian railway network and provided by RFI. RFI provided the access to the data of 5 months (January to May 2020) of train movements of one medium-sized railway station of the Liguria region, situated in the North West of Italy. A subsection of the station is shown in Figure 1; the station has been anonymised, and the complete structure can not be provided due to confidentiality issues. For the same reason, train identifiers have been anonymised. The modelled station includes 130 track segments (out of which 34 are track switches), 107 itineraries, 10 platforms, 3 entry points, and 3 exit points. The average number of trains per day was 130 before COVID-19-related lockdown and 50 after movement restrictions were enforced in Italy. As described in the corresponding section, the PDDL+ models provide a fully ground description of the problem to solve. The size of the considered problems, represented as the sum of actions, processes and events, ranges from approx. 300 to $1,200$.

The travel times information needed by the PDDL+ model, e.g., the time needed by each train to complete a track segment, to leave a platform, maximum times, etc., were calculated by leveraging on historical data provided by RFI. The whole dataset of historical tracks' travel times were clustered by a variety of features, such as train characteristics (e.g. passengers, freight, high speed, intercity, etc.), station transit characteristics (i.e. overall train trip inside the rail-network, and entry and exit points), weekdays, and weather conditions. According to the characteristics of the problem at hand, it is then possible to estimate the travel times of every train by assigning the average value of the times inside the corresponding cluster.

All the tests were run on a 2.5GHz Intel Core i7 Quad-processors with 16GB of memory made available and a Mac OS operating system. The cutoff time was set to 5 CPU-time minutes, but plans are usually generated in less than 30 seconds.

Validation of generated plans consisted of: (i) comparison with what would be expected in a common sense solution, by the visual inspection of the generated plans by a domain expert, and (ii) automated validation of the generated plans using the planner's internal validator.

Evaluation Scenarios

We considered three evaluation scenarios, for assessing the capability of the approach to: (i) accurately model historical data; (ii) minimise delays, and (iii) increment the railway station capacity.

Validation Against Historical Data. As initial test, we investigated the accuracy of the proposed PDDL+ formulation for modelling historical data. In other words, we are interested in assessing if the PDDL+ formulation is capable of representing how the addressed in-station train dispatching problem is currently faced by the human operators. This step is fundamental for at least two reasons: (i) to ensure that the PDDL+ formulation is realistic and can capture all the nuances of the real-world application, and (ii) to support the use of historical plans as baseline for validation and comparison purposes. To perform this analysis, we selected the day – in February 2020, before the start of the COVID-19 lockdown in Italy – with the minimum mean squared deviation of recorded train timings from the official timetable. This was done to guarantee that no emergency operations were executed by the operators. We encoded the recorded happenings of that day under the form of a single PDDL+ plan, using the operators introduced in the corresponding section. We then successfully validated such a plan, using the ENHSP validator, against our PDDL+ model. Notably, the fact that the PDDL+ model can correctly model the real-world dynamics, implies that planning-based tools can be straightforwardly exploited, and the planning engine can provide an encompassing framework for comparing different strategies to deal with recurrent issues, and for testing new train dispatching solutions. This result already represent a significant leap forward for the state of the art of the application field.

Minimisation of Delays. The next step is to focus on the benefits that the proposed PDDL+-based approach can deliver when dealing with delayed trains. Here we performed two different sets of experiments. First, we assessed how the

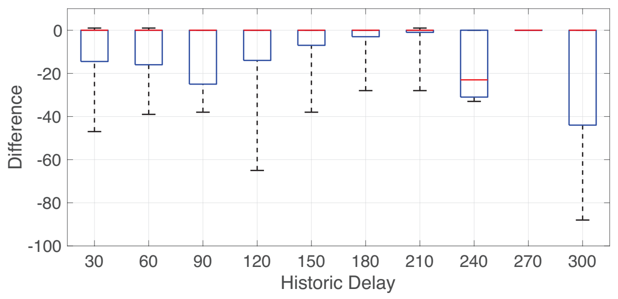

Figure 5: A box and whisker representation of the delay reduction achieved by our approach (y-axis), with regards to the recorded historical delay (x-axis). Whiskers refer to the highest and lowest reduction in delay, while the box (in blue) indicates the first, second, and third quartiles. Red indicates the median value. Negative y-axis values indicate that the planning approach reduced the delay, wrt historical records.

plans generated with our technique compares with the strategies currently implemented in the considered train station, in order to assess the improvement over the state of the art. Second, we performed an extensive analysis aimed at quantifying the ability of the approach to cope with increasingly large widespread delay. In order to deal with the first aspect, we considered data collected between January and March, prior to the COVID-19 related lockdown, and identified all cases where at least one train was delayed with regards to the official timetable. As for the planning scenario, we considered a time window centred on the delayed train(s) and bounded by the first time where no trains are in movement in the controlled station. What falls inside such time window is encoded in the considered planning problem. This allows to consider the entirety of the context in which the delayed trains have to operate, thus maximising the comparability of the results. In total, 311 scenarios including one or more delayed trains were identified using the described technique. Overall results are presented in Figure 5, where scenarios are clustered, in bins of 30 seconds, according to the average recorded delay (x-axis). The clusters' size vary between 2 (300 seconds delay) and 59 (30 seconds delay on average) instances; the number of trains per instance ranges between 1 and 12. Whiskers refer to the highest and lowest reduction in delay from the relative clustered recorded delay, while the box indicates the first, second, and third quartiles. Red is used to represent the median value. A value of 0 indicates that our approach performed exactly as the human operator in the recorded historical data; negative values on the y-axis indicate that we improved over the historical record. Results indicate that the PDDL+-based approach is never worse than the current strategy exploited at the controlled railway station, and that instead it is usually able to reduce the average delay. For instance, taking the clustered bin of 90 seconds, i.e. historical cases where trains were delayed by 90 seconds, our approach reduced the delay by up to 40 seconds. This is remarkable if we also take into account that the delay can be reduced only for trains that terminate or do an intermediate stop at the controlled station: there is no way to reduce the delay of trains that originate at the station – as they leave the platform late. By looking at the generated plans, our intuition is that the delay reduction is due to a better use of the available infrastructure, leading to shorter waiting times for trains entering the station.

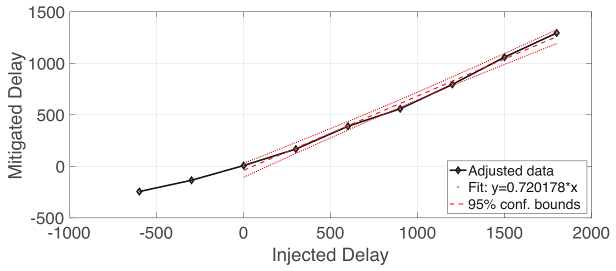

Figure 6: The relation between the injected Gaussian delay (x-axis) and the average mitigated delay obtained by exploiting the generated plans (y-axis), when considering the evening peak hours (17.00-20.30) on the controlled station.

To quantify the ability of our approach to handle delays we considered the peak hours time slots for the controlled railway station: (i) 06.30-09.30, with 24 moving trains, (ii) 12.00-14.30 with 17 trains, and (iii) 17.00-20.30 with 29 trains. (i) and (iii) include the commuters trains, while (ii) has a large number of trains used by students going home from school. Focusing on these three time slots, we selected the day, before the COVID-19 travel restrictions were put in operation, with the minimum mean squared deviation of recorded timings from the official timetable to be sure to have a levelled ground for performing the analysis. After that, we delayed every train of the considered time slot using a Gaussian-distributed delay $N(\mu, \sigma)$ with an increasing μ and a fixed σ of 5 minutes. μ ranged from -10 to $+30$ minutes, with 5 minutes steps.[3] For each value of μ, we randomly generated 10 planning instances using a Monte Carlo approach to inject delays to all the trains of the instance. Average results over the 10 instances are then considered. Figure 6 shows the achieved performance, as the relationship between the injected delay and the final observed average delay, on the time slot 17.00-20.30. Results on the other time slots are analogous. The behaviour can be interpolated by a straight line with an angular coefficient of 0.72; in other words, our proposed approach is able to plan the movement of trains to absorb 28% of the injected delay. For example, considering an injected Gaussian delay of 500 seconds, the final measured average delay is of 350 seconds: the proposed approach is able to absorb almost two minutes of the initial delay. These results are of particular relevance since in the considered scenarios all the trains are delayed (which is a worst case scenario) and the approach is still capable of reducing the average delay by almost a third.

Increment of the Railway Station Capacity. For this scenario, we conducted a stress test with the aim of understanding if the proposed approach can lead to an increment of the

[3]In our analysis, a negative delay is used to model a train that is early with regards to the official timetable.

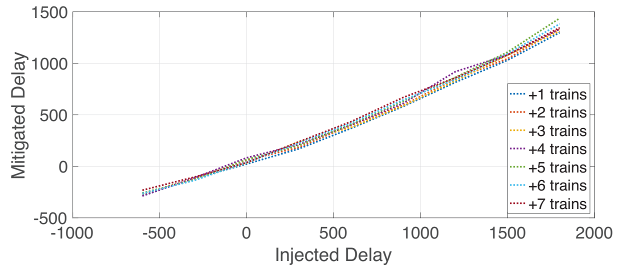

Figure 7: The relation between the injected Gaussian delay (x-axis) and the average mitigated delay obtained by exploiting the generated plans, in the presence of an increasingly large number of additional trains, when considering the evening peak hours (17.00-20.30) on the controlled railway station.

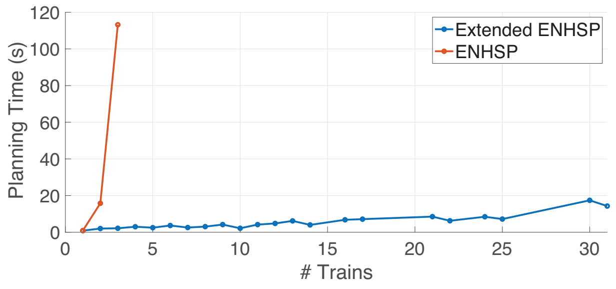

Figure 8: The CPU-time needed by the domain-independent version of ENHSP and the version extended with the domain-specific optimisations (Extended ENHSP) for solving instances involving an increasing number of trains.

railway station capacity. Using the same settings exploited for the previous scenario, we add an increasingly large number of synthetically generated trains as follows. Considering historical data, we randomly selected a train that originates, terminates, or does an intermediate stop at the station. The arrival/departure time of this synthetically generated train is then scheduled in between of existing trains in the time slot. After that, the usual Gaussian distributed delay (as described in the previous paragraph) is injected. This process is repeated 10 times. Then, another train is synthetically generated and added restarting the above mentioned procedure. The results of this set of experiments are presented in Figure 7. Surprisingly, results presented in Figure 7 are very similar to those shown in Figure 6: the fact that the station is serving up to 25% more trains does not result in a reduced capacity of our approach of mitigating the injected delay. Consequently, the proposed approach seems to be very robust. When more than 7 trains are added in the considered time slot, the limited availability of entry/exit points and platforms makes impossible to generate in-station dispatching plans that allows to satisfy even very loose constraints on the maximum time a train is allowed to stay in the station, or to wait at an entry point. Even a visual inspection performed by RFI experts, and some attempts to manually generate some reasonably timed dispatching plans, did not lead to the generation of any sensible solution. This suggests that our approach is able to exploit the available infrastructure up to a very high level, possibly close to the physical limit of the railway station.

Importance of the Domain-specific Extension. Finally, a natural question that we want to answer is: what is the impact of the designed domain-specific extensions on ENHSP? For this purpose, we need to quantify the performance improvement that can be obtained by using the introduced domain-specific optimisations considering planning tasks involving an increasing number of trains to be controlled. Focusing on the evening peak hour, we start by considering a time window where a single train is arriving at the station. We then incrementally increase the size of the considered time window, in order to include a larger number of trains at every step, up to the point where all the evening peak hour is modelled.

Figure 8 shows the results of the performed analysis in terms of runtime needed by the systems to generate a single solution. The domain-independent version of ENHSP is not able to solve instances that involve more than 3 trains, within the given cutoff time of 300 seconds. On the contrary, the domain-specific extensions we introduced allow the planning engine to quickly solve challenging instances that involve all the trains managed by the modelled railway station during the evening peak hour. All the instances, also those considered for the other scenarios of this experimental analysis, are solved in less than 20 CPU-time seconds when the proposed domain-specific extension is in operation. Looking at the importance of the extensions, a preliminary analysis suggests that both the specialised heuristic and the adaptive delta have a comparably strong impact on performance, but what gives a significant boost is using them together. Constraints are helpful, but not as much.

Conclusion

In this paper we presented an automatic solution to the in-station train dispatching problem. We modelled the problem in PDDL+, and designed a set of domain-specific enhancements that allow the ENHSP planning engine to quickly solve large and complex instances. Results on real-world historical data of a medium-sized railway station from North-West of Italy, provided by RFI, show the potential of our approach on a wide range of scenarios. In particular, the proposed approach demonstrated the ability to reduce delays and to better exploit the available infrastructure – with the potential of allowing a station to serve a larger number of trains without the need for structural modifications.

As future work, we plan to assess more in depth the importance of the domain-specific extensions, and to evaluate our approach on different and more complex railway stations. Moreover, we are interested in extending our approach to support timetable generation, and to address also the line dispatching problem.

Acknowledgements

This work has been partially funded by Hitachi Rail STS through the RAIDLab (Railway Artificial Intelligence and Data Analysis Laboratory), a joint laboratory between Hitachi Rail STS and University of Genoa. This work has been supported by RFI (Rete Ferroviaria Italiana) who provided the data for the analysis (we sincerely thank Renzo Canepa for his support). Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

Atzmon, D.; Diei, A.; and Rave, D. 2019. Multi-train path finding. In *Proceedings of SoCS*, 125–129.

Barrena, E.; Canca, D.; Coelho, L. C.; and Laporte, G. 2014. Exact formulations and algorithm for the train timetabling problem with dynamic demand. *Computers & Operations Research* 44: 66–74.

Billionnet, A. 2003. Using integer programming to solve the train-platforming problem. *Transportation Science* 37: 213–222.

Böcker, J.; Lind, J.; and Zirkler, B. 2001. Using a multi-agent approach to optimise the train coupling and sharing system. *European Journal of Operational Research* 131: 242–252.

Bryan, J.; Weisbrod, G. E.; and Martland, C. D. 2007. *Rail freight solutions to roadway congestion: final report and guidebook*, volume 586. Transportation Research Board.

Cacchiani, V.; Furini, F.; and Kidd, M. P. 2016. Approaches to a real-world train timetabling problem in a railway node. *Omega* 58: 97–110.

Cacchiani, V.; and Toth, P. 2012. Nominal and robust train timetabling problems. *European Journal of Operational Research* 219(3): 727 – 737.

Caprara, A.; Fischetti, M.; and Toth, P. 2002. Modeling and solving the train timetabling problem. *Operations research* 50: 851–861.

Caprara, A.; Galli, L.; Kroon, L.; Maróti, G.; and Toth, P. 2010. Robust train routing and online re-scheduling. In *Proceedings of ATMOS workshop*, 24–33.

Cardillo, D. D. L.; and Mione, N. 1998. k L-list λ colouring of graphs. *European Journal of Operational Research* 106: 160–164.

Chakroborty, P.; and Vikram, D. 2008. Optimum assignment of trains to platforms under partial schedule compliance. *Transportation Research Part B: Methodological* 42: 169–184.

Corman, F.; D'Ariano, A.; Pacciarelli, D.; and Pranzo, M. 2012. Bi-objective conflict detection and resolution in railway traffic management. *Transportation Research Part C: Emerging Technologies* 20: 79–94.

Cushing, W.; Kambhampati, S.; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proceedings of IJCAI*, 1852–1859.

D'Ariano, A.; Pranzo, M.; and Hansen, I. A. 2007. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *IEEE Transactions on intelligent transportation systems* 8: 208–222.

D'ariano, A.; Pacciarelli, D.; and Pranzo, M. 2007. A branch and bound algorithm for scheduling trains in a railway network. *European journal of operational research* 183: 643–657.

Fox, M.; and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *J. Artif. Intell. Res.* 27: 235–297.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based Policies for Efficient Multiple Battery Load Management. *J. Artif. Intell. Res.* 44: 335–382.

Kumar, R.; Sen, G.; Kar, S.; and Tiwari, M. K. 2018. Station Dispatching Problem for a Large Terminal: A Constraint Programming Approach. *Interfaces* 48: 510–528.

Lamorgese, L.; and Mannino, C. 2013. The track formulation for the train dispatching problem. *Electronic Notes in Discrete Mathematics* 41: 559–566.

Lamorgese, L.; and Mannino, C. 2015. An exact decomposition approach for the real-time train dispatching problem. *Operations Research* 63: 48–64.

Lamorgese, L.; Mannino, C.; and Piacentini, M. 2016. Optimal train dispatching by Benders'-like reformulation. *Transportation Science* 50: 910–925.

Lee, Y.; and Chen, C.-Y. 2009. A heuristic for the train pathing and timetabling problem. *Transportation Research Part B: Methodological* 43: 837–851.

Mannino, C.; and Mascis, A. 2009. Optimal real-time traffic control in metro stations. *Operations Research* 57: 1026–1039.

McCluskey, T. L.; and Vallati, M. 2017. Embedding Automated Planning within Urban Traffic Management Operations. In *Proceedings of ICAPS*, 391–399.

Ramírez, M.; Papasimeon, M.; Lipovetzky, N.; Benke, L.; Miller, T.; Pearce, A. R.; Scala, E.; and Zamani, M. 2018. Integrated Hybrid Planning and Programmed Control for Real Time UAV Maneuvering. In *Proceedings of AAMAS*, 1318–1326.

Rodriguez, J. 2007. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B: Methodological* 41: 231–245.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *Proceedings of ECAI*, 655–663.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.* 68: 691–752.

Scala, E.; and Vallati, M. 2020. Exploiting Classical Planning Grounding in Hybrid PDDL+ Planning Engines. In *Proceedings of ICTAI*, 85–92.