

# Intelligent Railway Platform Scheduling

*A thesis submitted*

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

**Sudhanshu Mishra**

*to the*

DEPARTMENT OF MECHANICAL ENGINEERING

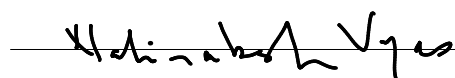
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

June, 2022



## CERTIFICATE

It is certified that the work contained in the thesis titled **Intelligent Railway Platform Scheduling**, by **Sudhanshu Mishra**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Prof Nalinaksh S Vyas

DEPARTMENT OF MECHANICAL ENGINEERING

IIT Kanpur

June, 2022



# ABSTRACT

Name of student: **Sudhanshu Mishra**      Roll no: **17807726**

Degree for which submitted: **Master of Technology**

Department: **MECHANICAL ENGINEERING**

Thesis title: **Intelligent Railway Platform Scheduling**

Name of Thesis Supervisor: **Prof Nalinaksh S Vyas**

Month and year of thesis submission: **June, 2022**

The present study aims to provide automated scheduling strategies for real-time train platforming at railway stations. In order to platform a train an inbound track, an outbound track and an arrival time has to be chosen such that it does not lead to conflicts with other trains. The station topology is given to the program as an undirected graph with nodes representing directions, platforms and shunting points of the railway tracks. The edges of the graph represent the railway tracks that connect the various resources of the station. Currently, trains are platformed manually by station controllers at each station that use previously made expert schedules. These schedules are prepared months in advance. However, in the case of delays, the controllers have to make decisions in real time. This usually leads to cascading effects delaying other trains, as they try to stick to the expert schedules. In this work, we use Mixed Integer Linear Programs to prepare efficient schedules to minimize delay due to platform or line unavailability. Furthermore, we have developed a virtual environment of the railway station to simulate different scheduling strategies and generate data. We use the data to create an intelligent agent which uses table based Q-learning to learn efficient platforming policies. To measure the effectiveness of each approach we measure the net delay that is produced using each strategy. We

use real station topology and schedules of Kanpur Central to compare our results.

Dedicated

To my Nana

Mr. Rama Kant Pandey

and my Nani

Mrs. Chanda Pandey





# Acknowledgements

I would like to extend my sincerest gratitude to my thesis advisor Prof. Nalinaksh S. Vyas for introducing me to this problem and guiding and supporting me in the right direction. I would also like to thank Dr. Vivek Sahai for providing me a copy of his work at the Observer Research Foundation on the Western Railway Suburban System timetable. His work was extremely insightful and provided me an in depth knowledge about the time-table generation process in the Railways. I would also like to thank my friends Kishan, Sagnik and Nikhil for their encouragement and discussions on my thesis. These discussions helped me a lot in figuring out solutions in my work. I would like to extend my gratitude towards my fellow classmates Sudhanshu Singh and Arihant Jain for helping me handle my TA duties during my time as a post-graduate student at the institute. I would also like to thank my labmates Anand Shekhar, Vandana Prakash and Sandeep for their advice and conservations during my time in the lab. I would like to thank my parents Mr. Krishna Shankar Mishra and Mrs. Pushpa Mishra for providing support and encouragement throughout my life. I would finally like to extend my utmost gratitude towards my brother Dr. Shubhanshu Mishra for his support and motivation towards research during my undergraduate and post-graduate days at IIT Kanpur.

Sudhanshu Mishra

Department of Mechanical Engineering, IIT Kanpur



# Contents

<b>List of Tables</b>	<b>xv</b>
<b>List of Figures</b>	<b>xvii</b>
<b>List of Figures</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Delays in Railway Schedules . . . . .	2
1.2 Delay Analysis . . . . .	4
1.3 Motivation . . . . .	7
1.4 Overview . . . . .	7
<b>2 Literature Review</b>	<b>9</b>
<b>3 Problem Statement</b>	<b>15</b>
3.1 Station Topology . . . . .	15
3.2 Train Data . . . . .	19
3.3 Schedule Data . . . . .	20
<b>4 Station Simulator</b>	<b>23</b>
4.1 Markov Decision Process . . . . .	23
4.2 Train Platforming as a Markov Decision Process . . . . .	25
4.2.1 State Representation . . . . .	25
4.2.2 Helper Functions and Data Structures . . . . .	26
4.2.3 Environment Simulation Dynamics . . . . .	30

4.2.4	Agent . . . . .	33
<b>5</b>	<b>Reinforcement Learning for Train Platforming</b>	<b>35</b>
5.1	RL for scheduling . . . . .	35
5.2	Value Function and Policy . . . . .	36
5.2.1	Bellman Equation . . . . .	36
5.2.2	Q-Learning Update Equation . . . . .	37
5.3	Agent Algorithms for TPP . . . . .	38
5.3.1	Deterministic Agent (Det) . . . . .	39
5.3.2	Random Agent . . . . .	39
5.3.3	Random Agent with Platform Schedule (Det-Random) . . . .	39
5.3.4	Q Learning Agent . . . . .	40
5.3.5	Reward Description . . . . .	41
<b>6</b>	<b>Mixed Integer Linear Programming Model</b>	<b>43</b>
6.1	MILP solver algorithms . . . . .	44
6.1.1	Branch and Bound . . . . .	44
6.2	TPP as a Mixed Integer Linear Programming Problem . . . . .	44
6.2.1	Pattern . . . . .	45
6.2.2	Pattern Incompatibility Graph . . . . .	45
6.2.3	Cost Function and Constraints . . . . .	48
<b>7</b>	<b>Experiment Results</b>	<b>51</b>
7.1	Case Study of Kanpur Central Station . . . . .	51
7.1.1	Assumptions . . . . .	52
7.1.2	Sunday . . . . .	53
7.1.3	Monday . . . . .	56
7.1.4	Tuesday . . . . .	59
7.1.5	Wednesday . . . . .	62
7.1.6	Thursday . . . . .	65
7.1.7	Friday . . . . .	68

7.1.8	Saturday . . . . .	71
7.2	Discussion . . . . .	74
<b>8</b>	<b>Conclusions</b>	<b>77</b>
8.1	Scope for further work . . . . .	78
	<b>References</b>	<b>81</b>



# List of Tables

6.1	Node Names and Patterns in the Pattern Incompatibility Graph . . .	47
7.1	Statistics of the algorithms on small instance of Sunday Schedule . .	54
7.2	Statistics of the algorithms on 24 hour instance of Sunday Schedule .	55
7.3	Statistics of the algorithms on small instance of Monday Schedule . .	57
7.4	Statistics of the algorithms on 24 hour instance of Monday Schedule .	58
7.5	Statistics of the algorithms on small instance of Tuesday Schedule . .	61
7.6	Statistics of the algorithms on 24 hour instance of Tuesday Schedule .	62
7.7	Statistics of the algorithms on small instance of Wednesday Schedule	63
7.8	Statistics of the algorithms on 24 hour instance of Wednesday Schedule	64
7.9	Statistics of the algorithms on small instance of Thursday Schedule .	66
7.10	Statistics of the algorithms on 24 hour instance of Thursday Schedule	67
7.11	Statistics of the algorithms on small instance of Friday Schedule . . .	69
7.12	Statistics of the algorithms on 24 hour instance of Friday Schedule . .	70
7.13	Statistics of the algorithms on small instance of Saturday Schedule .	72
7.14	Statistics of the algorithms on 24 hour instance of Saturday Schedule	73





# List of Figures

1.1	Platform Schedule with Delay while adhering to the initial schedule .	5
1.2	Platform Schedule with Delay and random replatforming in case of conflict while adhering to the initial schedule . . . . .	6
3.1	Example of a sample station layout . . . . .	17
3.2	Kanpur Central Station as seen on Google Earth . . . . .	18
3.3	Kanpur Central Station as seen on Google Maps . . . . .	18
3.4	Kanpur Central Station Graph Representation . . . . .	18
4.1	Agent Environment Interaction . . . . .	24
4.2	Rolled Out Representation of a Finite State MDP . . . . .	24
4.3	An example of the Path Maps used in the environment . . . . .	28
4.4	Layout of a test station describing the connections between the plat- forms and the incoming directions. . . . .	29
4.5	An example usage of the path incompatibility graph for the test station.	29
4.6	Flow Chart of Environment Dynamics . . . . .	32
5.1	An example of the Q-table used in the Q-learning agent. . . . .	41
6.1	An example of the pattern incompatibility graph for the test station for 3 trains. . . . .	46
7.1	Platform schedule for Sunday . . . . .	53
7.2	Comparison of Algorithms on small Sunday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	54

7.3	Comparison of Algorithms on 24 hour Sunday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . . . .	56
7.4	Platform schedule for Monday . . . . .	57
7.5	Comparison of Algorithms on Monday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	58
7.6	Comparison of Algorithms on 24 hour Monday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . .	59
7.7	Platform schedule for Tuesday . . . . .	60
7.8	Comparison of Algorithms on Tuesday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	60
7.9	Comparison of Algorithms on Tuesday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . . . .	61
7.10	Platform schedule for Wednesday . . . . .	62
7.11	Comparison of Algorithms on Wednesday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	63
7.12	Comparison of Algorithms on Wednesday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . . . .	64
7.13	Platform schedule for Thursday . . . . .	65
7.14	Comparison of Algorithms on Thursday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	66
7.15	Comparison of Algorithms on Thursday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . . . .	67
7.16	Platform schedule for Friday . . . . .	68
7.17	Comparison of Algorithms on Friday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	69
7.18	Comparison of Algorithms on Friday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . . . .	70
7.19	Platform schedule for Saturday . . . . .	71

7.20	Comparison of Algorithms on Saturday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule. . . . .	72
7.21	Comparison of Algorithms on Saturday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule. . . . .	73



# Chapter 1

## Introduction

The Indian Railways is the fourth largest railway system in the world by size. It has a route length of 67,956 km with about 80% of electrified routes as of 2022. It caters to 8 billion passengers and supports the transport of 1.2 billion tonnes of freight each year. The railway system is spread across the country covering 7,325 stations and currently has 13,169 passenger trains running on a daily basis. It offers services for both passengers and freight transport in the country. The rail network is ideal for bulk transportation of commodities and long distance travel along with being an economic and energy efficient mode of transportation.

In many places in India, local rail transport is an economic means of travel for the common man. Taking the case of Delhi Metro, more than 2 million people use it everyday for travelling to work and as an economic means for transport throughout the city. Another example is the suburban railways in Mumbai. The Western Railway and the Central Railways carry as many as 8 million commuters each day. Due to the rapid development and growth opportunities this number is going to increase in the coming years. Prior to the work of (Sahai et al. 2016), the Western and Central Railways followed a method of time-table construction dating to the days of Independence. However, these time-tables suffer from numerous shortcomings. They mention that there was no fixed platform for many trains at these stations and all these trains follow a tight schedule.

Scheduling is an important part of an efficient and reliable railway system.

The Indian Railway system is a massively complex entity which requires important scheduling and management decisions at different levels. On a macroscopic level, an optimal schedule has to be decided taking into account the whole network topology of stations and tracks in a region such that maximum number of trains can be scheduled. The more the number of trains scheduled results in increased passenger traffic and comfort, furthermore it leads to more profit for the railways.

## 1.1 Delays in Railway Schedules

It is quite common for trains to be delayed in India. These delays can be credited to a numerous reasons like weather, e.g., fog in winter months in north India and rains during summer monsoons countrywide and unavailability of platforms to name a few. In 2017-18, Indian Railways had the worst punctuality performance in three years. 30 per cent trains ran late in 2017-18, according to official data. There isn't enough space for trains on stations which is the main cause of delays, and little is being done to change that. The station development/redevelopment plans mainly address on facilities for the passengers on the station premises and facade of stations only and not on removing constraints and bottlenecks for ensuring timely arrival and departure of trains to/from the stations, which should be one of the most important parameters of the quality of service being provided to the passengers.

It is a primary requirement for inward trains to have a clear platform at their arrival. Due to non-availability of path or platforms, trains have to wait at the outer signal of adjacent stations until the platform is vacated by pre-occupied trains. The unavailability of platforms can be attributed to the occupation of other trains due to their stoppage at the station, non-availability of stabling/pit lines, late start of trains from the platform due to unavailability of locomotive, crew etc.

The report from the Comptroller Auditor General of India (Comptroller General 2018), explains how the unavailability of path (platform/line) for accommodating trains and absence of enough platforms with sufficient length to handle trains with 24 or more coaches are main reasons for delay of trains in Northern Railways. The

report points that one of the most important reasons for train delays was the lack of basic infrastructure which leads to detention of trains before they reach the station. In order to redevelop the performance of the railway system one of the points mentioned by the report is creation of a master plan for stations with heavy passenger traffic to identify constraints of station line capacity and devise measures to be taken to address these constraints on priority. It also suggested that the modernisation/redevelopment of stations should address infrastructural constraints and works such as construction of additional platforms, stabling and washing pit lines, remodelling of yards etc. should also be included in the scope of modernisation/redevelopment. During March 2017, between adjacent stations to the Kanpur Central station, there was en route detention of 47121 minutes in respect of 2851 trains. During March 2017, there were excess stoppage of 29813 minutes in respect of 2970 trains on platform at Kanpur Central station.

The report states that goods trains were significantly more delayed than passenger trains with the delays ranging from 20 to 100 minutes. Many passenger trains were stopped for more than 10 to 17 minutes beyond their stoppage times at New Delhi, Kanpur Central, Allahabad Junction, Bhopal and Howrah. Many passenger trains started late by 15 to 74 minutes at most of the stations. The maximum delays reported for a particular case was in the range of 100 to 165 minutes. The stations with significantly higher detentions were New Delhi, Kanpur Central, Allahabad Junction, Patna, Mathura and Vijaywada. They attribute this detention to the non-availability of clear line or path at the outer signal of the adjacent station. In more than 32% cases the reason for detention was the non-availability of clear path or traffic. During March 2017, 4248 passenger trains suffered a net delay of 77,989 minutes for 54 adjacent stations of the 15 stations selected for the audit. On 38 adjacent stations 7853 trains were detained due to the non availability of the path or platform at 11 stations.

One case that is observed quite frequently at the Kanpur Central Station (CNB) is that of long delays of Inter-City Trains. These trains suffer minimal delays reach-

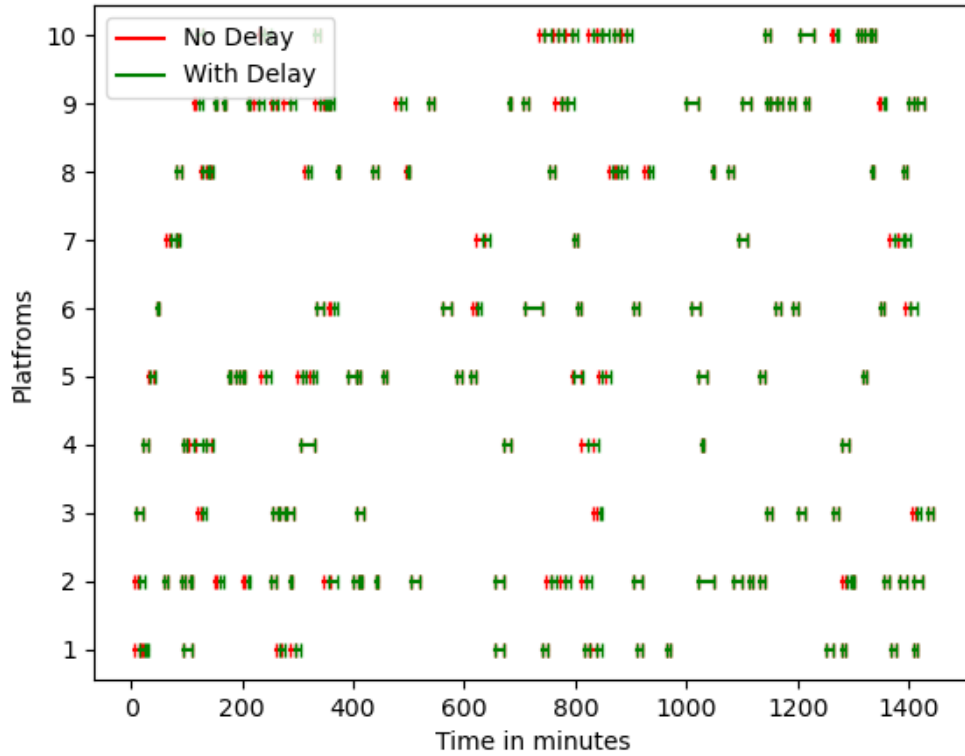
ing Chakeri station (station before CNB), however, it is quite common for these trains to park at the outer signal of CNB for hours due to the unavailability of platforms at the station. Platform schedules are made well in advance for stations and they require expert planners who work on it for months by analyzing the train routes and delay data to create a robust schedule. However, the final platform allocation is still controlled manually at each station trying their best to stick to the nominal time-table and platforms. This results in increased delay for future trains and inconvenience for the passengers. Every minute a train is delayed, it accounts for time wasted for all of the passengers in the train.

In the next section, we show a simulation of how the delays effect the final schedule while trying to respect the original time-table.

## 1.2 Delay Analysis

In this section, we create a simulation to observe the net delays that are produced in the time-table when some trains are subjected to random perturbations and the station controller still tries to follow the original platform schedule. We take the Kanpur Central Station (CNB) with 156 trains scheduled to arrive in a span of 24 hours. The station has 10 platforms and 3 directions for the arrival of trains to the station. We have randomly selected 30 trains out of 156 and have delayed them by 10 minutes. The initial scheduled is designed in such a way that if all trains stick to the schedule there will be 0 delay in total schedule. In order to schedule trains in the station we use a lock and release strategy, i.e. when a platform, arrival and departure path is selected, we lock the selected resources as well as all other resources that have some station resource in common with them. This is done to prevent the collision of two trains if they happen to meet at some common point in their paths.

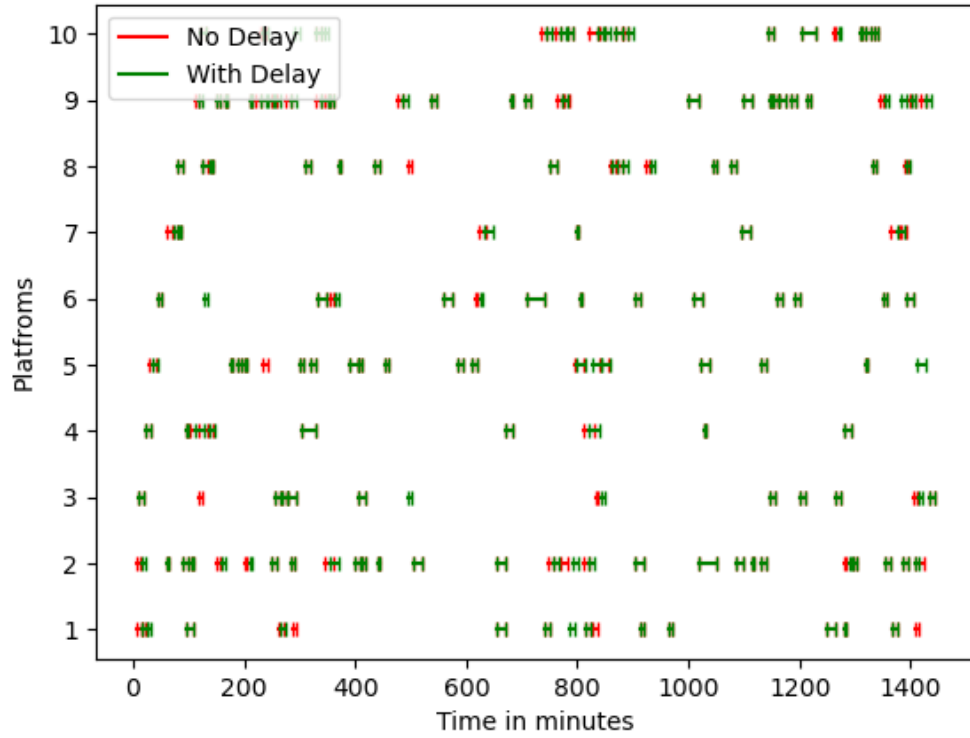




**Figure 1.1:** Platform Schedule with Delay while adhering to the initial schedule

In 1.1, the x-axis shows the time of the day in minutes from midnight. One day has 24 hours which is equal to 1440 minutes in a day. The y-axis shows the platforms in the station. Each data point on the plot shows the time interval for which the platform was occupied. The red data-points correspond to the initial schedule which results in 0 delay in the schedule for the delay. The green data-points correspond to the schedule generated when some trains have been delayed but the trains are still platformed according to the initial platforming schedule, that is they take the same arrival and departure paths as the initial schedule. The net delay induced into the schedule while still following the initial platforming strategy is **204** minutes.

In the next simulation, we induce the same delay into the initial schedule but in case the original platform is not available, we randomly choose an available platform, arrival and departure path. If no possible arrival or departure path is available, the train will get delayed.



**Figure 1.2:** Platform Schedule with Delay and random replatforming in case of conflict while adhering to the initial schedule

In 1.2, the x-axis shows the time of the day in minutes from midnight. One day has 24 hours which is equal to 1440 minutes in a day. The y-axis shows the platforms in the station. Each data point on the plot shows the time interval for which the platform was occupied. The red data-points correspond to the initial schedule which results in 0 delay in the schedule for the delay. The green data-points correspond to the schedule generated when some trains have been delayed but the trains are still platformed according to the initial platforming schedule, that is they take the same arrival and departure paths as the initial schedule. However, if the original platform is not available we randomly select another applicable platform for the train. The net delay induced into the schedule while still following the initial platforming strategy is **130** minutes.

### 1.3 Motivation

In this work, we focus on rescheduling the trains within the station, i.e. after they reach the home signal and are waiting for the allocation of platforms. As we have seen in 1.2, even for small delays like 10 minutes for some trains, the whole time-table of the station gets disturbed leading to delays of more than 3 hours. Minimizing this delay is of substantial importance for the railways as it leads to more profit for the railway management and less inconvenience for the passengers. As can be seen from the audit report of the Comptroller General of India, platform or line unavailability is an important cause of delays in many stations in Northern India like Kanpur Central and Allahabad Junction. The delays produced due to uninformed platforming are then propagated to the next station in line and thus would lead to similar effects on the subsequent station. We wish to study the platforming strategy on complex multi-direction multi-platform station and provide better scheduling strategies to mitigate this effect. We aim to provide an environment to simulate train routing in a station. These simulators can then also be used to test different scheduling strategies very quickly and can help planners to develop better heuristics and platform schedules. Another important usage of the simulator is to provide a general environment to test different reinforcement learning algorithms. There are many such environments that are created for different applications to motivate research and make comparison of different solutions easy. Some examples are the OpenAI gym for Atari games, flatland for research in train routing, Mujoco for robotics applications, etc. We also provide two scheduling agents to solve the train platforming problem.

### 1.4 Overview

In this work, we develop a station simulator that can be used for any station based on their station topology to simulate train routing through a particular station. We further discuss two approaches to platform trains in a station. Our first approach is a reinforcement learning based agent that can quickly learn and find a better

platforming strategy than selecting paths randomly in case of a delay. Our second approach models the problem as an Mixed Integer Linear Program and uses graph based techniques to provide a better solution.

# Chapter 2

## Literature Review

Most work related to train scheduling is in the area of network planning and scheduling trains on a network of stations. (Cordeau et al. 1998) provides an overview of the methods and models used in rail transportation. They give a summary of the methods used in the freight car management models, routing trains across a network, train dispatching models and locomotive assignment models. This work however does not put much focus on delays related to unavailability of platforms in a station but focuses on the scheduling of trains over a large railway network. (Lusby et al. 2011) also provides a good review of the current railway track allocation methods and models. They present a literature review for single track railway networks, routing trains at junctions and general railway networks. They provide an overview about conflict graph approaches like node packing and graph coloring, constraint programming and heuristic approaches for train routing. (Carey and Carville 2003) points out that most work done in train timetabling does not concern multi-platform stations where platform allocation will become a major issue. They present a heuristic algorithm that assigns each train a suitable platform, arrival and departure time and then loops back on the proposed schedule to resolve conflicts. They have designed their algorithm to produce results similar to that provided by manual train planners. (Cardillo and Mione 1998) present the problem of platform as a graph coloring problem. (Billionnet 2003) solves the problem formulated by (Cardillo and Mione 1998) using integer programming. Their works shows that integer program-

ming can be used for solving large instances of the platforming problem. (Ghoseiri et al. 2004) provide a multi-objective optimization model that focuses on reducing the fuel consumption and minimizing the passenger travel time which is similar to minimizing the train delay. They use a Pareto frontier detailed multi-objective optimization algorithm. However, they do not consider stations complex stations that receives trains from multiple directions. They do not show the results of their algorithm on a real world station but only on small numerical examples. (Zwaneveld et al. 2001) transforms the platforming problem into a node bin packing problem. They focus on minimizing the number of shunting operations and maximizing the platform preference of each train in the schedule. They solve the bin packing problem using integer linear programming. The work done by (Chakroborty and Vikram 2008) closely resembles our problem statement. They focus on optimizing the delay of the trains, allocation of preferred platforms and minimizing last minute platform changes. They focus on solving the problem at a station for a window of 120 minutes.

(Peter Sels et al. 2014) presents an interesting outlook of the platforming problem from the perspective of the infrastructure management company. They provide a MILP model to optimize the platforming problem for ten Belgian Stations. (Bai et al. 2014) provide a MILP model to generate a conflict free time-table by considering both continuous and discrete time domain models. The objective function used by them focuses on minimizing the line occupancy rate at the station. The largest problem size they consider is of 60 trains. (Akyol 2017) provides an analogy between the train platforming problem and the parallel machine scheduling problem. They show that the platforming problem is atleast NP-complete in nature. They propose a meta-heuristic algorithm which considers platform track assignment, calculation of total delay and improving the algorithm by eliminating delays. (Pellegrini et al. 2014) study railway traffic management in the railways after perturbations to certain operations. They seek to find the best train routing strategy and schedule in the case of perturbations. They study the impact of interlocking systems in train management in the Lille Flandres station and Gagny area of France. They use

mixed integer programming for determining the train routing strategy.

(Cacchiani et al. 2014) provide an amazing tutorial on train scheduling for people who are new to the field. They provide a MILP based model for generating a schedule for the non-periodic train timetabling problem, i.e. scheduling trains over a railway network for the non-periodic case. They also provide a detailed description of modelling the train platforming problem for a particular station using 0-1 binary integer programming. Our integer linear programming approach is inspired from the tutorial. They present a solution for an artificial station with two incoming and outgoing directions and use a quadratic model for computation. The tutorial is based on (Caprara et al. 2007a) work for train platforming. They use a general quadratic integer programming model for platforming trains and focus on minimizing the number of platforms used, minimizing the train delay and the time of soft incompatibilities. They propose the usage of a pattern incompatibility graph to model the platforming problem. In the paper (ibid.) also propose a method to effectively linearize the quadratic part of the loss function. They compare their results with the currently used heuristic at the Rete Ferroviaria Italiana and provide results for three Italian stations.

(D’Ariano et al. 2007) consider the railway scheduling problem as a variant of the job-shop scheduling problem having no-wait and blocking constraints. They provide an analogy to the job shop scheduling problem where a track corresponds to a machine and a train corresponds to a job. They use a branch and bound algorithm to solve the scheduling problem after converting the original problem in an alternative graph formulation. They present the results of their algorithm on the Schiphol railway network in the Netherlands. However, they do not analyze the platform scheduling problem at the station level but are concerned with rescheduling trains on the railway network. (Dewilde et al. 2013) consider complex busy stations having limited capacity for delay propagation. They present a framework for routing trains, timetabling and platforming. They use Tabu search for their timetabling phase, a MILP based approach for the routing phase and a heuristic approach for

the platforming phase. They consider two Belgian stations for their case study and also discuss a method to quantify the robustness of their approach. (Dessouky et al. 2006) propose a rule based branch and bound algorithm for optimal dispatching times of trains in complex rail networks. They consider route length and train speeds while making the platform scheduling decisions. They present their results on a portion of the Los Angeles railway network. (Tornquist and Persson 2005) proposes a tabu search and simulated annealing based approach for train scheduling over a network in order to minimise train delays. They present experiments dealing with the Swedish railway network.

(Corman et al. 2009) propose an efficient algorithm for determining resource conflicts and delays in railway operations. They study the impact of delay propagation between trains on the railway timetable. They study the effect of delays based on real data on the Swiss Railway Network. (Sahai et al. 2016) provide a detailed analysis about improving the Indian Western Railways scheduling problems. They provide a detailed analysis of the overcrowding of local trains and change in the number of services to improve the state of the Western Railways. They propose a SARAL timetable which has been carefully designed by observing the real-time data to improve train kilometers and passenger comfort. Their leads to an increase of 165 trains in the local train network and reduction of overcrowding in suburban trains. It reduces the need for passengers to cross tracks to get to the desired platforms in case of sudden delays and platform rescheduling leading to fewer accidents and loss of human life on the tracks. The proposed time-table has ramped-up the net distance travelled by trains in the schedule by 6,371 kilometers. This work is an excellent showcase of the amount of expertise and time needed by expert planners to improve railway scheduling problems.

Recent papers in this area have started applying machine learning techniques for scheduling problems in the railways however, the work in the area of train platforming is still scarce. The most prominent work related to our problem statement is by (Salsingikar and Rangaraj 2020). They provide a reinforcement learning based



approach for train movement planning at a station. They present a brief description about their state-space representation and the environment used for training the RL agent. To train the agent they use the On-Policy Monte Carlo algorithm as mentioned in (Sutton and Barto 2018). They use a route-lock, route-release operation policy for modelling the problem. In order to present their results they use the CSMT Harbor terminal station which has 2 directions for arrival and departure of trains. There are other papers which use machine learning for railway traffic management however, they are not associated with the problem of platforming but still they give us good insights about the problem formulation for such techniques.

(Prasad et al. 2021) present a policy search approach for railway scheduling using covariance matrix adaptation evolution strategy (CMA-ES). They aim to produce schedules weighted delay optimised schedules for long railway lines consisting of multiple stations. They present their results on Kanpur, Ajmer, KRCL and 2 hypothetical railway line instances. (Khadilkar 2019) proposes a scalable reinforcement learning approach for track allocations on a train given their initial positions, halt and traversal times while minimizing the priority weighted delay. They use a modified epsilon-greedy policy using q-values for determining the policy of their agent. (Šemrov et al. 2016) presents a Q-learning based approach for train scheduling for single track railway stations. They present a method to formulate the state space of the environment. They showcase their results on a real-world railway track in Slovenia consisting of 23 block sections, 14 stations and 26 trains. Hiroshima (2011) presents a reinforcement learning system for generating marshaling plan of freight cars in a train. They minimize the total distance that a locomotive has to travel in order to obtain the desired layout for an outbound train. They use Q-learning for deciding on the action policy for their agent. (Ning et al. 2019) propose a deep reinforcement learning approach for minimizing the average delay of all trains in a railway line. They focus on the Beijing-Shanghai high-speed railway line. They use a Deep Q Network (DQN) for the agent policy network. They show that they were able to reduce the average total delay by 46% in the case of disturbance scenarios.

Train Scheduling and platforming bears some resemblances to job-shop scheduling problems. There has been a lot of work in the machine learning community to solve job-shop instances which can be used to gain insights into using ML for train scheduling problems as well. (S. C. Riedmiller and M. A. Riedmiller 1999) present a reinforcement learning approach to learn local dispatching policies for job-shop scheduling. They give a formulation of the job-shop problem as a markov decision process and use a q-learning based algorithm for policy updates. They present their results on two benchmarks in job-shop scheduling. (Runarsson et al. 2012) propose monte carlo tree search and pilot methods for learning efficient strategies for dispatching jobs in the job-shop scheduling problems. They give a detailed overview of the algorithm and its usage in scheduling. They compare their results with a branch and bound algorithm by (Brucker 2001) on relatively small benchmark problems ranging from 6 X 6 to 20 X 20 in size.

# Chapter 3

## Problem Statement

Our problem statement deals with assigning each incoming train to a station a platform and a suitable arrival time. The departure time is calculated as the sum of the arrival time and the stoppage time of the train. In order to platform a train, we also need to assign an inbound path from the incoming direction to the platform and an outbound path from the platform to the outgoing direction. The selection of tracks is extremely essential as no two trains can take the same path at the same time to avoid accidental hazards. Another important thing to keep in mind is to have a headway time between two trains platformed trains on the same platform. This is done for safety reasons. The headway time for trains to specific platforms may be different. The objective is to assign each train a suitable platform such that the net delay caused due to the unavailability of a platform or line for the entire schedule is minimised. We will describe the terms associated and the data representation in the following section.

### 3.1 Station Topology

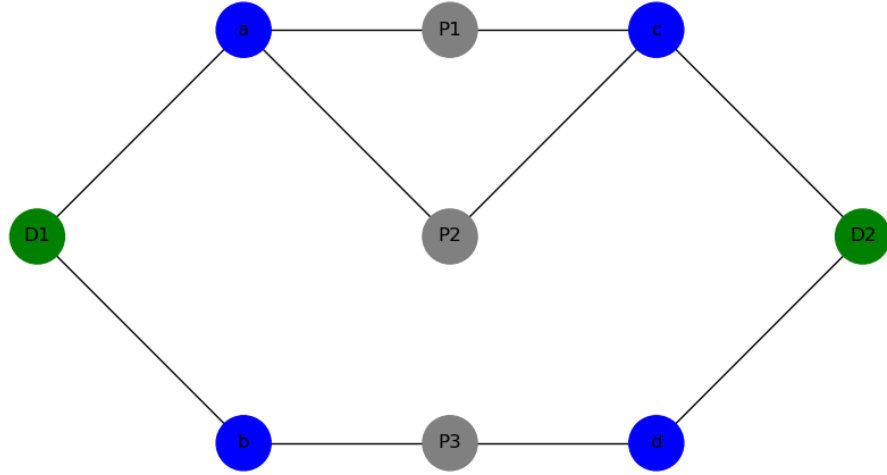
In order to schedule trains properly, we need to take into account the topology of the station. By station topology, we focus on the connectivity of different resources of the station with the tracks.

We represent each station as an undirected graph. Each station is represented with

the following elements : directions to the station, platforms, shunting points of tracks which form the nodes in the undirected graph. All the nodes are connected using edges which represent the physical tracks that join these elements of the station. Each edge is represented as a tuple of two nodes. Based on this layout we can calculate the inbound and outbound paths for the station. Each inbound path is represented as an ordered string of nodes, with the first node being a direction node corresponding to the incoming direction and the last node being a platform node. Each outbound path is represented as an ordered string of nodes, with the first node being a platform node and the last node being a direction node corresponding to the outgoing direction. From this ordered string it is easy to figure out the edges in the graph that correspond to these paths. All inbound and outbound paths are acyclic and directed.

It is important to notice that most paths share some nodes of the station. This leads to only some of the paths being used at a given instant of time. In order to ensure the safety of trains, no two paths having some node in common can be allotted at the same time. However, it should also be noted that direction nodes are a bit different from the other nodes. These nodes do not represent any physical entity in the station and are an abstraction which help define the boundaries of the station. Thus paths that only share direction nodes are not incompatible with each other. Another important thing to notice is that in this undirected graph representation, each inbound path will have a twin outbound path. However, it is not necessary for the station to have such symmetry in real life. Some paths can be specifically assigned to be only inbound or outbound as well.

Below we give an example of a sample station and its graph representation.



**Figure 3.1:** Example of a sample station layout

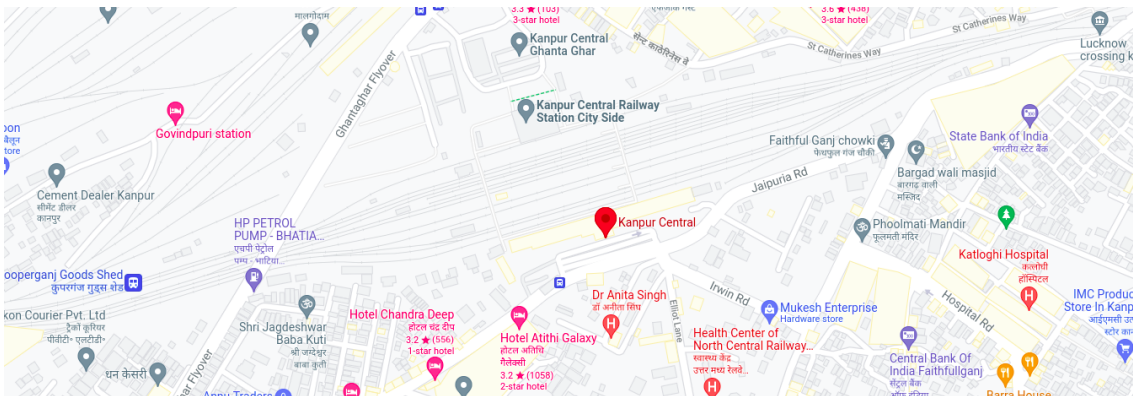
The nodes and edges in the graph are defined below:

- Nodes **P1**, **P2**, **P3** correspond to platforms 1, 2 and 3 in the station.
- Nodes **D1** and **D2** correspond to directions 1 and 2 to the station.
- Nodes **a .. e** correspond to the shunts connecting different railway tracks in the station.

Now we present a more realistic case of the Kanpur Central Station (CNB). Below we present an image taken from Google Earth of the station.

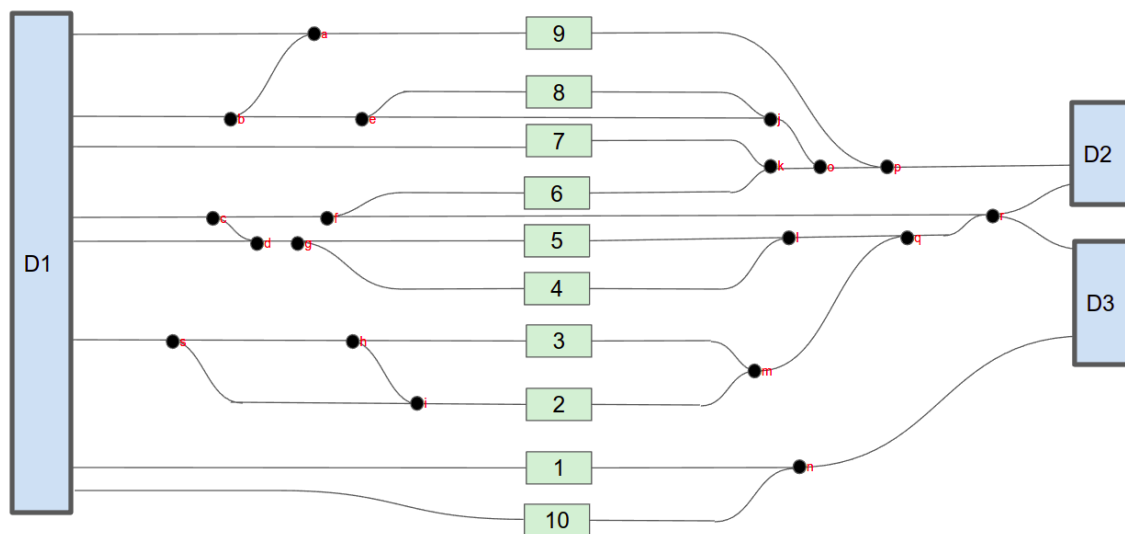


**Figure 3.2:** Kanpur Central Station as seen on Google Earth



**Figure 3.3:** Kanpur Central Station as seen on Google Maps

Using the above figures we were able to properly abstract out the graph representation of the Kanpur Central Station. Kanpur Central has trains coming from 3 different directions. It has 10 platforms and 17 shunting points across the station. Below we present the abstracted graph representation of the station.



**Figure 3.4:** Kanpur Central Station Graph Representation

The nodes and edges in the graph are defined below:

- Nodes **1 ... 10** correspond to platforms **1 - 10** in the station.
- Nodes **D1**, **D2** and **D3** correspond to directions 1, 2 and 3 to the station.
- Nodes **a .. r** correspond to the shunts connecting different railway tracks in the station.

## 3.2 Train Data

Each Train is characterised by the following attributes:

1. **Train Name** : Name of the Train
2. **Train No** : Number for identification of the train
3. **Arrival Time** : Expected Arrival Time of the train to the station. This is subject to change based on the time the train is actually scheduled on the station.
4. **Stoppage Time** : Time for which the train stops at the station.
5. **Departure Time** : Time at which the train leaves the station.   
Departure Time = Arrival Time + Stoppage Time
6. **Incoming Direction** : The direction from where the train enters the station.
7. **Outgoing Direction** : The direction in which the train leaves the station.
8. **Preferred Platforms** : The Preferred platforms for a Train. Many high priority trains usually have a set platform to which they arrive based on historic reasons.

### 3.3 Schedule Data

The train time-table for Kanpur Central has been taken from the cleartrip<sup>1</sup> website. There were numerous websites that provided the time table for singular trains however, a complete list of all trains stopping at Kanpur Central was not easily available. The cleartrip website contained a complete list of all trains that were stopping at Kanpur Central with the days at which each train was running with their arrival and departure times and their preferred platforms. However, in order to ascertain the incoming and outgoing directions we referred to each train's travel route and looked at Google Maps to figure out its travel direction which matched with our graph representation of the station. For example, the three main stations surrounding the Kanpur Central Station are the New Delhi Railway station, Charbagh Station in Lucknow and Allahabad Junction. These three stations matched with the three directions to the Kanpur station and hence the directions for most of the trains were assigned based on these stations. The stoppage time has been calculated as the difference between the departure and arrival time of each train. In the schedule that we consider, the number of trains running on each day are as follows:

- Sunday : 155 trains
- Monday : 155 trains
- Tuesday : 155 trains
- Wednesday : 152 trains
- Thursday : 157 trains
- Friday : 154 trains
- Saturday : 157 trains

However, the exact arrival and departure path of the trains in the station is not available online. We generate these paths for the nominal schedule by solving

---

<sup>1</sup><https://www.cleartrip.com/trains/stations/CNB>



each days schedule using the MILP model. The generated schedule is free from any conflicts on any tracks for all the trains. Thus, we use this stabilised schedule as our platform schedule for our experiments.



# Chapter 4

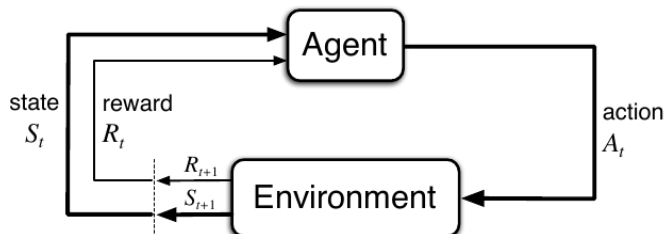
## Station Simulator

We have created a real time station simulator environment to turn the train platform problem into a Sequential Markov Decision Process (MDP). This step is quite important as without that, the problem becomes quite complex when we have to make multiple decisions at a given point of time. Different permutation of decisions would lead to different results in a scheduling problem. Most Reinforcement Learning algorithms require the problem to be stated as an MDP before a suitable algorithm can be used to solve it.

### 4.1 Markov Decision Process

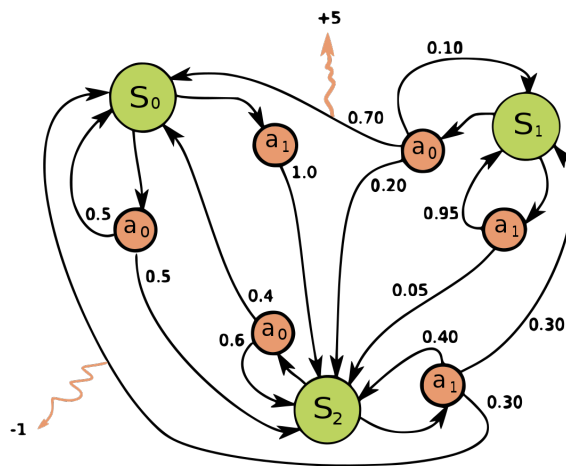
Markov decision processes are a way to frame the problem of learning of an agent while interacting with an environment to reach an objective [Sutton and Barto 2018](#). The decision making component is called the agent. The thing with which it interacts, gets observations and rewards, is called the environment. The environment usually has some state variables. Some or all of the state variables may be visible to the agent. The states that are visible to the agent are called the observations. The general framework is, the environment generates observations which are then processed by the agent. The agent then using some decision making algorithm selects an action and passes it to the environment. The environment transitions to the next state based on the action received and its internal dynamics. This state

transition results in a reward. The reward is basically a signal to the agent telling it how good or bad its action was. The state transition dynamics of an environment can be deterministic or stochastic. Below we present a general agent environment interaction in an MDP.



**Figure 4.1:** Agent Environment Interaction

MDPs are classified into two categories depending upon whether they have a terminal state or not. MDPs that have a terminal state are called episodic MDPs and those without one are called non-episodic MDPs. We present below a general rolled out version of the dynamics of a finite state MDP. The figure has been taken from (Sutton and Barto 2018).



**Figure 4.2:** Rolled Out Representation of a Finite State MDP

Mathematically an MDP is represented as a tuple  $(S, A, P_a, R_a)$  where :

- $S$  is a set of states called the state space.
- $A$  is a set of actions called the action space.

- $P_a$  is the probability of transitioning from state  $S_t$  to  $S_{t+1}$  under the action  $a$
- $R_a$  is the reward received by taking action  $a$  at state  $S_t$

The agent and environment interact with each other at time steps  $t = 0, 1, 2, 3, \dots$

This interaction gives rise to a trajectory :

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots$$

The agent updates its policy on the basis of the rewards it receives in each trajectory.

The aim of each agent is to maximum the net sum of discounted rewards ( $G_t = R_{t+1} + \gamma R_{t+2} \dots$ ).

## 4.2 Train Platforming as a Markov Decision Process

As discussed in the previous section, we need to define an agent and an environment for the MDP. The environment in this case is the station and the agent is the platform controller of the station which makes the decision where to platform each train. The environment consists of all the platform, the resources of the station and all the trains passing through the station. We now describe the state representation of the platform.

### 4.2.1 State Representation

The state space of the station consists of 6 vectors. They are described below:

- **in\_path** : It is a bit vector of length  $N$  where  $N$  is the no. of arrival paths in the station which has been computed previously. When a path is occupied or locked, its value is 1 otherwise it is 0.
- **out\_path** : It is a bit vector of length  $M$  where  $M$  is the no. of departure paths in the station which has been computed previously. When a path is occupied or locked, its value is 1 otherwise it is 0.
- **pfs** : It is a bit vector of length  $P$  where  $P$  is the no. of platforms in the station which has been computed previously. When a platform is occupied,

its value is 1 otherwise it is 0.

- **in\_path\_t** : It is a vector of length  $N$  where  $N$  is the no. of arrival paths in the station which has been computed previously. When a path is occupied or locked, its value is equal to the time required for the path to get unlocked. When a path is not locked its value is 0.
- **out\_path\_t** : It is a vector of length  $M$  where  $M$  is the no. of departure paths in the station which has been computed previously. When a path is occupied or locked, its value is equal to the time required for the path to get unlocked. When a path is not locked its value is 0.
- **pf\_t** : It is a vector of length  $P$  where  $P$  is the no. of platforms in the station which has been computed previously. When a platform is occupied, its value is equal to the time required for the platform to get unlocked. When a platform is unoccupied its value is 0.

Before we describe the state transitions in the environment, we will first present some important helper functions and data structures that help in efficiently performing these transitions.

## 4.2.2 Helper Functions and Data Structures

In order to maintain order of incoming trains and updates of station resources we use two types of priority queues, train queue and message queue.

### Priority Queue

A priority queue is a min heap that is used to store data in the First In First Out (FIFO) order similar to a queue. However, we can additionally apply a priority order to our data to additionally arrange them according to their priority. The priority queue provides  $O(1)$  lookup of the minimum element whilst providing  $O(\log(n))$  complexity for insertion and deletion operations. It is an efficient data structure to process job queues in scheduling problems.

## Train Queue

In order to keep track of the order of incoming trains, we use a priority queue with trains having a priority on their arrival times. The train with the earliest arrival time is closer to the top of the queue and served earlier than those that arrive later. There is an additional priority check for those trains that have the same arrival time. Each train has a priority number assigned to it. We have a 3 level priority order as follows:

- 1 : For historically important trains like Rajdhani.
- 2 : For superfast trains like Duronto, Humsafar, etc.
- 3 : For all other trains.

Trains with a lower priority order will be served earlier when two trains have the same arrival time.

## Message Queue

To automatically unlock station resources like platforms and tracks when they are available we use messages. A message is an object that stores the id of the resource to be unlocked and the clock time when it has to be unlocked. to properly keep track of all of the messages, we use a priority queue with messages that need to be executed earlier having higher priority than others.

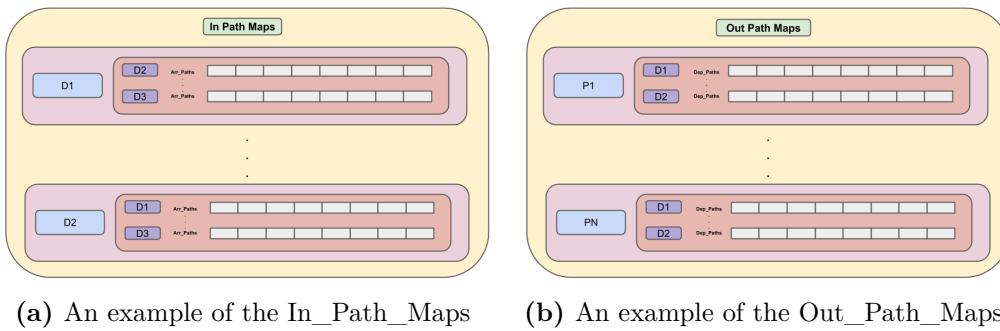
## Path Maps

To have fast accesses to paths from a direction we preprocess all paths at the start of the simulation. We create two nested python dictionaries to store the connectivity of the platforms and paths. The path maps make searching for paths during the simulation extremely fast. The two path maps are described below:

- **in\_paths\_from** : The keys are the incoming directions and the values are themselves a dictionary with keys being the outgoing directions and the values being the inbound paths for the pair of directions.

- **out\_paths\_from** : The keys are platforms and the values are dictionaries whose keys are outgoing directions and their values are those outbound paths that start at the given platform and end at the outgoing direction

In Fig 4.3, we give an example of the two path maps used in the environment. These path maps can be augmented to contain extra information about the arrival and departure paths, like the platform capacity, degree of connectivity of the path, no. of shunting nodes, etc.



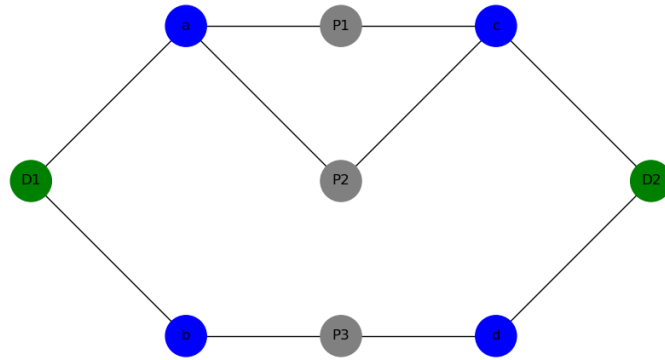
**Figure 4.3:** An example of the Path Maps used in the environment

### Path Incompatibility Graph

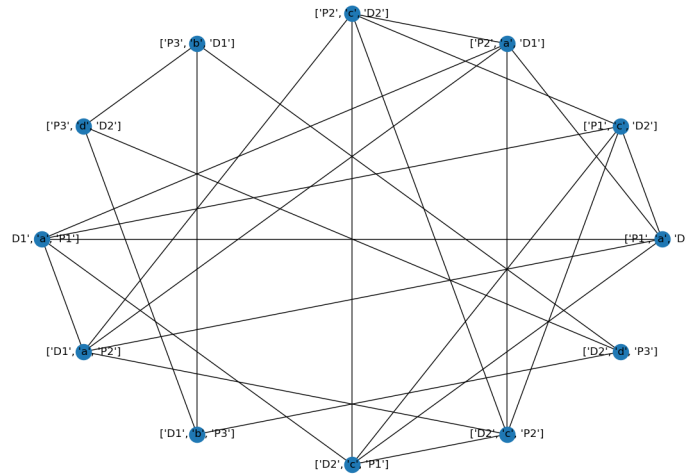
An incompatibility graph connects those nodes that are incompatible with each other. In our case, each node corresponds to an inbound path or an outbound path. The incompatibility constraint between two nodes is as follows : **if two paths share at least one resource (a platform or an intermediate shunting point in the station) except the direction of the path then they are said to be incompatible with each other.** This is an extremely important constraint from the safety of the trains. Consider the case where two paths sharing a resource are simultaneously chosen for two trains. This would then result in the two trains using the same resource at the same time. In reality, this may lead to a collision between the trains or perhaps a deadlock on the station. The most efficient way to resolve this problem would be to monitor the shared resource and allow one train to pass while the other is waiting. This solution is a bit more complex and requires careful implementation. Thus for this study we simply lock incompatible paths once a train



has been scheduled. The path incompatibility graph is extremely useful when we need to lock additional paths that would become unsuitable when a given train is scheduled. Using the path maps and the path incompatibility graph makes the search time for possible paths extremely quick thus leading to faster simulation run-time. Below we give an example of the path incompatibility graph for a test station and its usage :



**Figure 4.4:** Layout of a test station describing the connections between the platforms and the incoming directions.



**Figure 4.5:** An example usage of the path incompatibility graph for the test station.

In Fig 4.4, we present the layout of a test station and in Fig 4.5 we present the path incompatibility graph for the station. The nodes of the graph are both the arrival and departure paths of the station. Between any two paths that share atleast some resource (platform or a shunting node), there exists an edge. Thus, all paths that are incompatible with each other are connected with an edge in the path incompatibility graph. This graphs helps in locking incompatible paths of a selected path. The incompatible paths can be queried quickly from the graph by looking at the neighbors of the selected path. We do not show the path incompatibility graph for the Kanpur Central Station as it has 50 nodes and so many edges that it becomes difficult to comprehend anything in the image.

### 4.2.3 Environment Simulation Dynamics

The environment steps through in a loop and updates its states while interacting with the agent when required. Before starting the simulation for the first time, we need to initialize some environment variables. We first need to initialize the station topology, the platforms, the arrival and the departure paths of the station from the station configuration files. We then need to compute the path maps and create the path incompatibility graph. We then need to initialize two important environment variables before each simulation run. They are defined as follows:

- **clock\_time** : The clock time is the actual time at the station. The least count of the clock time is in minutes. Each day has 24 hours which is equal to 1440 minutes. It remains in the range  $[ 0, 1440 )$  and is stored as an integer for easy usage. We can easily convert it into whichever time format we require using python's datetime library. The clock time is useful for tracking the various events in the simulation like platforming trains and unlocking resources.
- **step\_time** : The step time is the simulation run time. As the simulation runs, it is updated by one unit. It remains in the range  $[ 0 , \text{max-steps} )$ . Usually clock time is not equal to the step time. As we have converted our

problem into a sequential MDP, we can only make one decision at a given point of time. However, it is very common for multiple decisions to be made at a given time in train scheduling. For example, it can happen that there are two or more trains that can be platformed at a given time. Thus, we would only need to update the step time of the environment after making one decision and then transition to the next state. We only update the clock time when there are no more decisions left to be made at that particular time step.

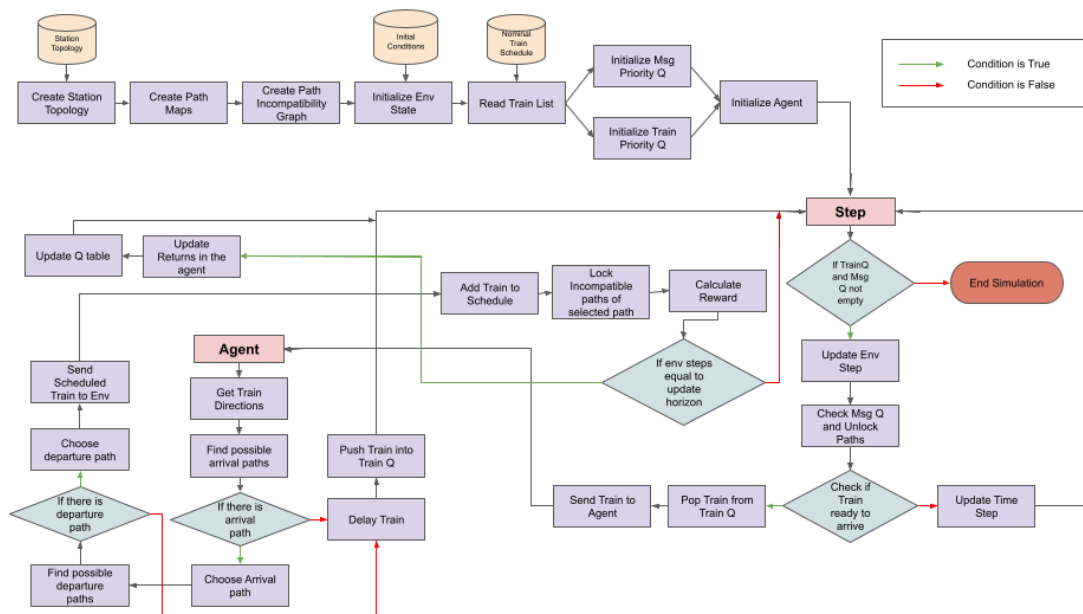
Before starting a simulation we need to initialize the train queue with the estimated arrival times of the trains passing through the station. We then need to initialize the state of the environment corresponding to the actual state of the station at the starting clock time. We can initialize the message queue as well by already locking the occupied paths and providing adequate messages to free them in the future. We then need to initialize the agent and pass it to the environment.

### **Simulation Loop**

The environment loops through steps until it reaches max-steps or until there are no more trains in the train queue and no more messages in the message queue. In each iteration of the loop, it first checks the loop termination condition, if it is false it then checks the message queue and unlocks the required paths and updates the state accordingly. It then compares the arrival time of the topmost train in the train queue and if it is equal to the clock-time, then we have a train that is ready to be platformed. It pops the train from the train queue and passes it to the agent along with the current state of the environment. The agent then using its decision making algorithm can either platform the train or in the event that no suitable platform is present it can signal the environment to delay the train. In the event of a delay, the arrival time of the train is incremented by  $t$  units ( we take  $t = 1$ ) and also updates the reward for that train and pushes it back into the train queue. It is important to note that as the train queue is a priority queue, the train will not be pushed to the end of the queue but according to its arrival time in the train

queue. If the agent has decided to platform the train, then the environment updates the reward of the train and locks the platform, inbound and outbound paths of the train and pushes appropriate messages for unlocking these paths in the future into the message queue. We then use the path incompatibility graph and create a list of all the neighbors of the chosen paths in the graph. These neighbor paths are incompatible with the chosen arrival and departure paths and thus they are also required to be locked. As we are using the path incompatibility graph, this search only takes  $O(\text{degree}(\text{path}))$  time and thus it is more efficient than searching the paths in the station. After making the decision, the scheduled train is added to the environments *trajectory* variable. This keeps track of the schedule made and can also be used later for updating the agents weights. Finally, we update the step-time by 1. When there is no action to be taken at a given step i.e. there is no train to be scheduled at the current clock-time, then we update both the step-time and clock-time. This is done so as to take multiple decisions at a given time-step and also obey the sequential decision making of our MDP.

Below we provide a flow chart of the environment dynamics :



**Figure 4.6:** Flow Chart of Environment Dynamics

#### 4.2.4 Agent

An agent using a specific algorithm has its own unique variables. However, there are some fundamental functions and variables that each agent possesses. We will describe them below:

- **act** : The act function is the core of each agent. It receives the environment state and the train that needs to be platformed. It first creates a list of possible in-paths from the incoming direction of the train using the path maps. It then sends the data to the choose-in-path function which returns an in-path using the agent's algorithm. If there are no possible in-paths available, then the agent signals the environment to delay the train. If an in-path has been decided, then we extract the platform from the given in-path. The platform will be the last node in each in-path. Based on the platform, the agent then calculates the possible out paths using the out\_paths\_from path map. It then sends the data and the list of possible out paths to the choose-out-paths function which returns an out-path. If there are no possible out-paths then the agent signals the environment to delay the train. If an out-path has been decided then the agent returns a tuple of four values ( train, in-path, out-path, platform, delay ). In the case of a delay, the delay value is 1 in the returned tuple and the three resource values are *None*.
- **choose-in-path** : This function takes the environment state, train and the list of possible in-paths as input. It decides upon which in-path out of the possible in-paths is to be selected and then returns the id of that in-path.
- **choose-out-path** : This function takes the environment state, train and the list of possible out-paths as input. It decides upon which out-path out of the possible out-paths is to be selected and then returns the id of that out-path.
- **update** : The update function updates the weights of the agent. It is different for different algorithms.



## Chapter 5

# Reinforcement Learning for Train Platforming

In the previous chapter, we described the real time station simulator and how it transforms the train platforming problem into a sequential MDP. We will now focus on reinforcement learning algorithms to solve our problem. Each RL algorithm usually focuses on developing a value function for the different states in the environment. The agent then creates its policy using the estimated value function taking actions that maximize the net return (discounted sum of rewards) of the trajectory. The algorithm tries to estimate the value function and then updates the agents policy. We will now define what mean by value function and policy.

### 5.1 RL for scheduling

In this section we describe some of the instances where reinforcement learning has been used to create better decision making agents which motivated us to use RL for solving our problem. There has been active research in creating intelligent agents to play games like chess, go, TD-Gammon and even more complex video games like Dota and Starcraft. Games present complex decision making problems where the agent has to adapt to the changing environment and create smart strategies to defeat its opponent. There are many success stories of RL agents excelling in

games. (Tesauro 1995) was one of the first RL agents that achieved a strong intermediate level of play in the game TD-Gammon. In the recent years, with use of DeepRL techniques agents were created by (Silver et al. 2017) which displayed superhuman level of play in games like chess and shogi. These achievements motivated researchers to create smart RL agents for complex decision making problems like job-shop scheduling (Zhang et al. 2020), vehicle routing (Nazari et al. 2018) and train line scheduling (Khadilkar 2019). There has been increased interest in developing environments and simulators to simulate these problems and then train agents in simulation to learn efficient strategies, one such example is flatland (Mohanty et al. 2020). This motivated us to think of train platforming as a reinforcement learning problem.

## 5.2 Value Function and Policy

The value function is an estimate of the net return that the agent can get from that state in the MDP. An important thing to note here is that it is for a state in an MDP. Another formulation of the value function is the Q-value function which is an estimate of the net return that the agent can get from that state taking a specific action. Thus, essentially the value function of a state can be calculated as an expectation of all the Q-values for that state.

The policy of an agent is the way that the agent decides on an action in a state. The policy can be stochastic or it can be deterministic. An agent creates its policy by taking the action that maximizes the value of that state. We will now define the Bellman Equation that defines the update rule for the value function and how it is used to update the policy of an agent.

### 5.2.1 Bellman Equation

The Bellman equation in essence captures the relationship between the value of a state and its successor states. The Bellman equation for the value function is as



follows:

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi}(s')] \quad (5.1)$$

where  $v_{\pi}(s)$  is the value of state  $s$  under the policy  $\pi$ ,  $\pi(a|s)$  refers to the policy that the agent is following at state  $s$ ,  $p(s',r|s,a)$  is the probability of transitioning to the state  $s'$  and getting reward  $r$  from state  $s$  by taking action  $a$ ,  $r$  is the reward,  $\gamma$  is the discount factor and  $v_{\pi}(s')$  is the value of the next state.

To get the optimal value for each state, the Bellman Optimality Equation is used:

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \quad (5.2)$$

where  $v_*(s)$  is the maximum value of  $v(s)$  according to the current estimate,  $a$  is the action taken in the current state that achieves the maximum  $v(s)$ ,  $p(s',r|s,a)$  is the probability of transitioning to the state  $s'$  and getting reward  $r$  from state  $s$  by taking action  $a$ ,  $r$  is the reward,  $\gamma$  is the discount factor and  $v_*(s')$  is the optimal value of the next state.

The policy is updated according to the action chosen in the bellman optimality equation update.

### 5.2.2 Q-Learning Update Equation

The q-learning algorithm has a Q-function that maps state-action pairs to expected returns.

$$Q: S \times A \rightarrow \mathbb{R} \quad (5.3)$$

Initially before training the agent, the  $Q$  function is initialized to an arbitrary value, in our case 0. Then, at each time the agent selects an action  $a_t$  observes a reward  $r_t$  and enters a new state  $s_{t+1}$  from previous state  $s_t$ , the value of  $Q(s_t, a_t)$  is updated. The  $Q$  update equation is similar to the Bellman update equation and is as follows:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (5.4)$$

where  $r_t$  is the reward received when transitioning from state  $s_t$  to  $s_{t+1}$  while taking action  $a_t$ ,  $\alpha$  is the learning rate (01) and  $\gamma$  is the discount factor.

It is important to note that  $Q^{new}(s_t, a_t)$  is the sum of three factors namely :

- $(1 - \alpha)Q(s_t, a_t)$  : this part of the update corresponds to the current value of  $Q(s_t, a_t)$ . This term in the update equation signifies how much of the previous data to retain in the  $Q$  state. For  $\alpha$  closer to 1, the update mostly focuses on the immediately received reward and disregards the previously stored information. For such values the updates to the equation are more rapid.
- $\alpha \cdot r_t$  : This term in the update signifies how much of the current received reward  $r_t$  to pass onto the  $Q$  value. For  $\alpha$  closer to 1, the update gives more weight to the currently received reward. In the extreme case of 1, the update equation becomes a greedy update equation, prioritizing immediate rewards.
- $\alpha \cdot \gamma \max_a Q(s_{t+1}, a_t)$  : This term signifies the maximum return that can be obtained from the state  $s_{t+1}$ . This part of the equation contains information of the future rewards and the effect that the current action had on the future states.

For all terminal states  $s_{end}$ , the update equation becomes :

$$Q(s_{end}, a_t) = \alpha \cdot r_t \quad (5.5)$$

### 5.3 Agent Algorithms for TPP

In this section, we will describe the algorithms used by the agents in the real time train simulator.

### 5.3.1 Deterministic Agent (Det)

This agent possess the predefined platform schedule for the station. At each step it assigns the train the path and the platform mentioned in the platform schedule. In case of delay or unavailability of path or platform, it just delays the train and waits until the specified path and platform is available for the train.

This agent is similar to the station controller following the platform schedule at all times.

### 5.3.2 Random Agent

The random agent has the same configuration as the base agent described in 4.2.4. In order to platform a train, it first finds the possible arrival paths according to the arrival and departure direction using the path maps. The random agent then randomly selects one of the possible paths. It then repeats the same steps for finding the departure path. If at any point of time, the no possible path is available the agent delays the train.

The random agent is akin to person trying to schedule a train without any knowledge of the station topology and previous data.

### 5.3.3 Random Agent with Platform Schedule (Det-Random)

This agent extends the random agent interface. It also possess the platform schedule for the station. At each step it assigns the train the path and platform mentioned in the schedule. In case of delay or unavailability of path or platform, it searches for alternative possible paths according to the arrival and departure direction of the train using the path maps. It then randomly selects one of the possible arrival path for the train. In accordance with the chosen arrival path a platform is selected. It then again checks for the possible departure paths in the path maps and selects one at random. If at any point it is unable to find an arrival or departure path of a train, it delays the train.

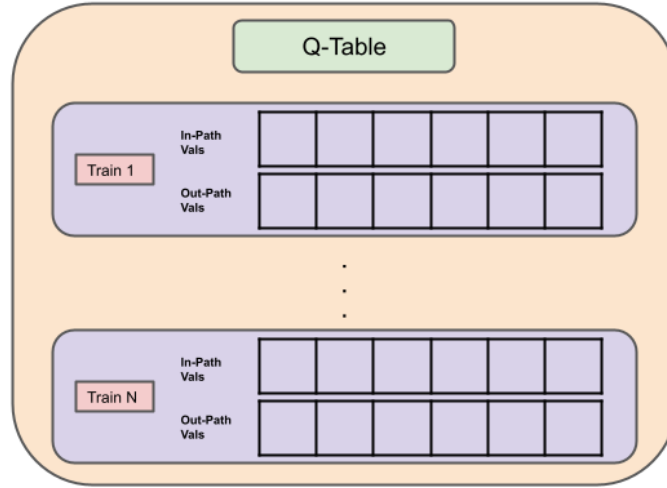
This agent is similar to the station controller following the platform schedule and in case of a delay, it tries to reschedule the train randomly.

### 5.3.4 Q Learning Agent

The Q-Learning Agent also has the same configuration as the base Agent with some additional features. It first has a variable called q-table which is the main update location for the q-learning algorithm. In order to platform trains the agent uses an epsilon-greedy strategy, i.e. with probability  $\epsilon$  that it takes the optimal action according to the stored q-values and with probability  $1 - \epsilon$  it takes a random action. During training we update the epsilon linearly w.r.t to the episodes starting with an initial value of 0 and ending with value 1. During the start of the training, the agent essentially samples actions randomly, but with time it gets a good approximation of the expected return of the different actions and starts to rely more on the q-values stored in the q-table. We will now describe how the q-table is maintained for this agent.

#### Q-Table

In the agent, the q-table is implemented as a dictionary with keys being the train names and the values being the q-values for the arrival and departure paths. For each train, we store two vectors with size equal to the number of arrival paths and the number of departure paths in the station respectively. For each arrival path chosen for a train, the index corresponding to that arrival path is updated with the estimated return for that state-action pair. Similarly, for each departure path chosen for a train, the index corresponding to that departure path is updated with the estimated return for that state-action pair. For each train we also maintain two more vectors in the train that track the frequency with which each action is selected. An example of the q-table used in the agent is given in 5.1.



**Figure 5.1:** An example of the Q-table used in the Q-learning agent.

### 5.3.5 Reward Description

For each train, there are two parts to the reward given. They are mentioned below:

- Delay Reward : For every minute a train is delayed a reward of -187 is given. Thus the net delay reward the train gets is  $-187 * (\text{minutesdelayed})$ .
- Platforming Reward : If a train is platformed with the preferred platform then it receives a reward of +89 otherwise it receives a reward of +55.
- Net Reward: The net reward received by the train is the sum of Delay Reward and Platforming Reward.

The Q-learning agent tries to maximize the expected return (sum of discounted rewards) for each train. Thus, in order to maximize the expected return it tries to minimize the delay reward and tries to choose the preferred platform for the train. The reward function is mainly dominated by the delay related reward however every time a train is platformed with the preferred platform, it essentially increases the likelihood of selecting the preferred platform. As the delay reward is much greater than the platforming reward, the agent's decision making will give more priority to minimizing the delay of the train. The values of the rewards are chosen in such a way so that delay would incur a very heavy penalty compared to platforming as

it is assumed that each train would get platformed eventually thus, each train is destined to get that reward at some time. The exact values are chosen randomly and other values can be tried to determine which values would give better results, out of the several values we chose during our experiments we found these to work the best however, the search is not exhaustive.

We have implemented various variants of the Q-learning algorithm that differ in either the way we update the returns in the Q-table or in some other implementation detail that was crucial to the algorithm. We discuss these variant below:

- **Update Horizon** : The update horizon is the number of steps of the environment after which we make the updates of the rewards to the q-table. We experimented with different values of the update horizon. Updating the rewards at the end of the episode led to bad results because there was a large amount of noise in the returns due to the epsilon greedy policy which led to poor convergence of the q-values. The q-values were not able to correctly record the delay signal for the actions. For extremely low value of update horizon like 10 we also did not get good results as then q-values for a train were not able to get a good idea about the action's effect on future trains and only focused on it's own rewards. To get the best results we need to select a sweet spot that gives a good approximation of the effects on the future as well as having very small noise. We found this sweet spot to be 100 environment steps.
- **Return** : We experimented with two types of returns. In one case, as the reward we sent the discounted sum of the rewards for each train in the update horizon. In the other case, it was just the net reward that the train got after being scheduled.
- **Mean Return** : In this variant, instead of using the q-update equation we updated the q-values by the weighted mean of the rewards using the frequency of the actions that are stored in the q-table.

# Chapter 6

## Mixed Integer Linear Programming Model

In this chapter, we describe the Mixed Integer Linear Programming (ILP) Model for train platform scheduling. MILP approaches are quite prevalent in scheduling problems. They have been used extensively in job shop scheduling, rolling stock problems and airline crew scheduling. The main idea behind this approach is to formulate the problem as a mixed integer linear program and specify the constraints required for a feasible solution. We then feed the integer linear program to a mathematical optimization solver like CBC, Gurobi etc. These solvers have been designed with advanced heuristics to solve such constrained optimization problems efficiently and quickly. Constrained optimization problems like job shop scheduling are NP-hard in nature, i.e. there currently doesn't exist a polynomial time algorithm to solve them. Usually, such problems have an exponential time complexity. Due to the advancement of hardware and more complex heuristic algorithms, these solvers are now able to solve even extremely large problems with millions of variables in a couple of minutes. However, the time taken to solve a particular problem is not directly dependent on the size of the problem but also the problem constraints. It has been observed that problems with just a couple hundred variables take hours to solve while extremely large problems with simpler constraints take only a few

minutes. This is partly due to the increasingly large search tree that is created in highly constrained problems. At the base, these solvers use the branch and bound algorithm to search the solution space. We will briefly describe the branch and bound algorithm in the next section.

## 6.1 MILP solver algorithms

We now describe some of the most popular algorithms that are used to solve constrained optimization problems.

### 6.1.1 Branch and Bound

The branch and bound is a tree search algorithm used for solving integer linear programs. It starts by relaxing the integer constraints on the variables and solving the resulting linear program which can be solved using any of the existing algorithms like simplex. We then take one of the variables whose value has to be an integer and then create two child nodes. One child will have the value of that variable equal to the floor of the previous solution while the other will have it as the ceiling. The two child nodes then again result into a linear program. If the cost of a child is above the feasible assignment value, then that tree is pruned otherwise the same branching is done at the child nodes. Finally, on reaching a leaf node, we backtrack to find the solution to the integer linear program.

## 6.2 TPP as a Mixed Integer Linear Programming Problem

As mentioned earlier, Train Platforming deals with a specific station. The station topology in this approach is the same as described in 3.1. We create a set of platforms, directions to the station, arrival and departure paths. We also have a schedule of trains coming to the station with their arrival, departure times, preferred plat-



forms and priority for each train. In this problem formulation we have two important variables for each train called **max-shift** and **granularity**. Max-Shift refers to the maximum delay that can be given to each train. If a train cannot be platformed within this maximum delay, then this train is not included in the final schedule. This is done to have a limited state space for the ILP formulation, otherwise the ILP would have an extremely large search space which will increase time it will take for the solver to find a solution. Granularity refers to the number of points to take in the interval  $(0, max - shift)$ . This is also used to reduce the state space of the problem. For example, the number of nodes in the pattern incompatibility graph for  $max - shift = 20$  and  $granularity = 2$  is 13530. To create the search space we create a pattern incompatibility graph. We will describe the pattern and the pattern incompatibility graph in the below.

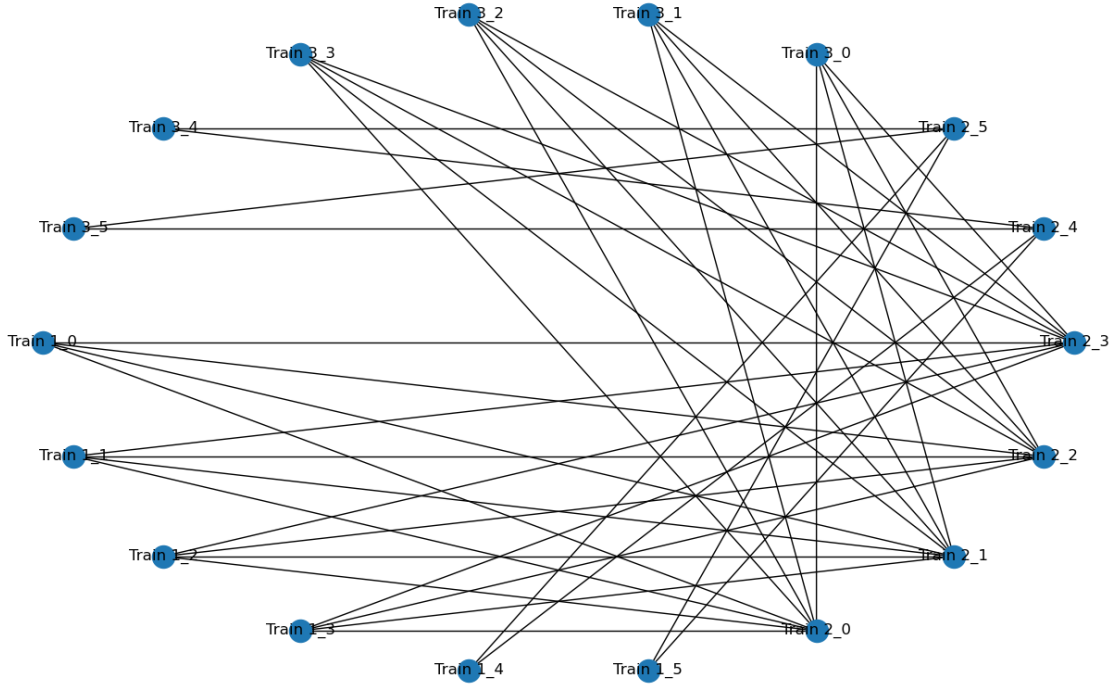
### 6.2.1 Pattern

A train is said to be platformed when it is assigned an arrival path to the station, a departure path from the station, a platform, an arrival time and a departure time. A pattern is defined as a tuple of these five values for a train. Thus, each train will have these five values assigned to it in the final schedule. In order to reduce, the search space we assume that the  $departure - time = arrival - time + stoppage$ , which results in essentially four independent variables that need to be found. For each train, all possible patterns are created i.e. all combinations of platforms, arrival paths, departure paths and possible arrival times within the max-shift and using the granularity.

### 6.2.2 Pattern Incompatibility Graph

The Pattern incompatibility graph has all the patterns for all the trains as its nodes. The most important part of the graph are its edges. Every pair of nodes that are incompatible with each other are connected with an edge in the graph. The incompatibility conditions are defined as follows:

- Two patterns occupying the same platform with an overlap between their arrival and departure times are said to be incompatible with each other.
- Two patterns having an overlap between their arrival and departure times and atleast one common resource in either the arrival or departure path (with the exception of the direction) are said to be incompatible with each other.



**Figure 6.1:** An example of the pattern incompatibility graph for the test station for 3 trains.

In the Fig 6.1, we give an example of the pattern incompatibility graph generated for the test station mentioned in Fig 4.4. In the Table 6.1, we describe the node names in the pattern incompatibility graph and the patterns associated with that node. Every pair of nodes that are temporally and spatially incompatible are connected with an edge in the graph. For example nodes Train 3\_0 and Train 2\_3 are incompatible as the shunting nodes *a* and *c* in the station are being occupied for the same instant of time. In the patterns, we additionally have mentioned the delay from the original arrival time of the train.

Thus, the train platforming problem reduces to selecting nodes in the pattern incompatibility graph such that there are no two nodes that share an edge with

Node name	Pattern
Train 1_0	('Train 1', 'P1', ['D1', 'a', 'P1'], ['P1', 'c', 'D2'], '09:02', '09:10', 0)
Train 1_1	('Train 1', 'P1', ['D1', 'a', 'P1'], ['P1', 'c', 'D2'], '09:04', '09:12', 2)
Train 1_2	('Train 1', 'P1', ['D1', 'a', 'P1'], ['P1', 'c', 'D2'], '09:06', '09:14', 4)
Train 1_3	('Train 1', 'P2', ['D1', 'a', 'P2'], ['P2', 'c', 'D2'], '09:02', '09:10', 0)
Train 1_4	('Train 1', 'P2', ['D1', 'a', 'P2'], ['P2', 'c', 'D2'], '09:04', '09:12', 2)
Train 1_5	('Train 1', 'P2', ['D1', 'a', 'P2'], ['P2', 'c', 'D2'], '09:06', '09:14', 4)
Train 1_6	('Train 1', 'P3', ['D1', 'b', 'P3'], ['P3', 'd', 'D2'], '09:02', '09:10', 0)
Train 1_7	('Train 1', 'P3', ['D1', 'b', 'P3'], ['P3', 'd', 'D2'], '09:04', '09:12', 2)
Train 1_8	('Train 1', 'P3', ['D1', 'b', 'P3'], ['P3', 'd', 'D2'], '09:06', '09:14', 4)
Train 2_0	('Train 2', 'P1', ['D2', 'c', 'P1'], ['P1', 'a', 'D1'], '09:05', '09:15', 0)
Train 2_1	('Train 2', 'P1', ['D2', 'c', 'P1'], ['P1', 'a', 'D1'], '09:07', '09:17', 2)
Train 2_2	('Train 2', 'P1', ['D2', 'c', 'P1'], ['P1', 'a', 'D1'], '09:09', '09:19', 4)
Train 2_3	('Train 2', 'P2', ['D2', 'c', 'P2'], ['P2', 'a', 'D1'], '09:05', '09:15', 0)
Train 2_4	('Train 2', 'P2', ['D2', 'c', 'P2'], ['P2', 'a', 'D1'], '09:07', '09:17', 2)
Train 2_5	('Train 2', 'P2', ['D2', 'c', 'P2'], ['P2', 'a', 'D1'], '09:09', '09:19', 4)
Train 2_6	('Train 2', 'P3', ['D2', 'd', 'P3'], ['P3', 'b', 'D1'], '09:05', '09:15', 0)
Train 2_7	('Train 2', 'P3', ['D2', 'd', 'P3'], ['P3', 'b', 'D1'], '09:07', '09:17', 2)
Train 2_8	('Train 2', 'P3', ['D2', 'd', 'P3'], ['P3', 'b', 'D1'], '09:09', '09:19', 4)
Train 3_0	('Train 3', 'P1', ['D1', 'a', 'P1'], ['P1', 'c', 'D2'], '09:13', '09:18', 0)
Train 3_1	('Train 3', 'P1', ['D1', 'a', 'P1'], ['P1', 'c', 'D2'], '09:15', '09:20', 2)
Train 3_2	('Train 3', 'P1', ['D1', 'a', 'P1'], ['P1', 'c', 'D2'], '09:17', '09:22', 4)
Train 3_3	('Train 3', 'P2', ['D1', 'a', 'P2'], ['P2', 'c', 'D2'], '09:13', '09:18', 0)
Train 3_4	('Train 3', 'P2', ['D1', 'a', 'P2'], ['P2', 'c', 'D2'], '09:15', '09:20', 2)
Train 3_5	('Train 3', 'P2', ['D1', 'a', 'P2'], ['P2', 'c', 'D2'], '09:17', '09:22', 4)
Train 3_6	('Train 3', 'P3', ['D1', 'b', 'P3'], ['P3', 'd', 'D2'], '09:13', '09:18', 0)
Train 3_7	('Train 3', 'P3', ['D1', 'b', 'P3'], ['P3', 'd', 'D2'], '09:15', '09:20', 2)
Train 3_8	('Train 3', 'P3', ['D1', 'b', 'P3'], ['P3', 'd', 'D2'], '09:17', '09:22', 4)

**Table 6.1:** Node Names and Patterns in the Pattern Incompatibility Graph

each other. This would result in a feasible solution to the problem. An additional objective in the problem is to achieve a feasible solution and have the minimum cost. In order to convert this problem into an integer linear program, we assign to each pattern a binary variable  $x_t^p$  where  $t$  refers to the train and  $p$  refers to the pattern that is associated with it. The variable  $x_t^p$  can either have a value of 0 signifying the pattern is not part of the final solution or a value of 1 indicating that it has been selected in the final schedule.

### 6.2.3 Cost Function and Constraints

In the cost function, we focus on three things : minimizing the delay, platforming as many trains as possible and providing trains with their preferred platforms. Thus, the cost function is defined as follows:

$$C = c_1 + c_2 + c_3 \quad (6.1)$$

where  $c_1, c_2, c_3$  are defined as follows:

$$c_1 = -\alpha \sum_{t \in T} \sum_{p \in P_t} x_t^p |d_t^p| \quad (6.2)$$

$$c_2 = \beta \sum_{t \in T} \sum_{p \in P_t} x_t^p \quad (6.3)$$

$$c_3 = \gamma \sum_{t \in T} \sum_{p \in P_t} x_t^p w_t^p \quad (6.4)$$

where  $\alpha, \beta$  and  $\gamma$  are parameters to scale the importance of each cost,  $d_t^p$  refers to the delay associated with pattern  $x_t^p$  and  $w_t^p$  is a binary variable whose value is 1 if the pattern  $p$  associated with train  $t$  has the preferred platform for train  $t$ . Here  $x_t^p$  refers to the binary variable of train  $t$  with pattern  $p$ .  $P_t$  refers to the set of patterns associated with the train  $t$  and  $T$  refers to the set of trains that have been scheduled for arrival to the station. It is important to note that  $w_t^p$  is not a decision variable and its value is set depending on the pattern associated with the train. The MILP is solved to maximise the cost function. Thus,  $c_1$  is the part of the cost

function that tries to minimize the delay. When we try to maximize  $c_2$ , it results in more number of patterns being selected, hence more trains are platformed. The cost function  $c_3$  is responsible for providing the preferred platform to each train. This cost function is subject to the following constraints:

**Constraint 1 :** For  $\forall$  trains  $t \in T$

$$\sum_{p \in P_t} x_t^p \leq 1 \quad (6.5)$$

In the above constraint,  $T$  refers to the set of all trains scheduled for arrival to the station and  $x_t^p$  refers to the binary variable of train  $t$  with pattern  $p$ . This constraint makes the solution have only one pattern selected for each train.

**Constraint 2 :** For  $\forall$  edges  $\in G$

$$x_{t1}^{p1} + x_{t2}^{p2} \leq 1 \quad (6.6)$$

In the above constraint,  $G$  refers to the pattern incompatibility graph,  $x_{t1}^{p1}$  and  $x_{t2}^{p2}$  refers to the binary variable of train  $t1$  with pattern  $p1$  and train  $t2$  with pattern  $p2$ . This constraint forces the solver to select at most one node from an edge in the Pattern Incompatibility Graph.

For solving our MILP, we maximize the cost function and use the CBC solver. All experiments involving this approach have been solved on a Lenovo Legion Y520 Laptop with 16 GB RAM. For experiments involving around 150 trains and max\_shift of 20 with a granularity of 2, the program takes almost 13 minutes to create the pattern incompatibility graph and takes an additional 5 to 15 minutes to provide a feasible solution.



# Chapter 7

## Experiment Results

In this section, we provide a case study of our algorithms on the Kanpur Central Station. We take the station topology as described in 3.1 and provide an analysis of the delays produced on a perturbed ideal time-table of the CNB station and the effects of rescheduling trains using the presented algorithms.

### 7.1 Case Study of Kanpur Central Station

In this case study, we take an initial ideal time-table (that produces 0 delays if all trains are on time) for each day of the week and perform a delay analysis for the following instances :

- A small instance of 40 trains in the original schedule where all trains have been perturbed by a random delay ranging from  $[10,55]$  minutes.
- A full 24 hour instance of the schedule where 70 random trains have been perturbed by a random delay ranging from  $[10,55]$  minutes.

We generate 40 different schedules for each perturbation and analyze the delay produced by each algorithm. We consider the horizon of 24 hours and try to platform as many trains that can be platformed within the 24 hour time-period after providing the perturbations.

### 7.1.1 Assumptions

We have taken the following assumptions while performing the experiments in this case study :

- for trains starting and ending at the station we assume a stoppage time of 15 minutes on the station.
- for all schedules we assume, a headway of 1 min between two consecutive trains on the same platform.
- As we did not have information about the washing lines and the yard lines for the station, we assume an appropriate arrival or departure direction for the trains starting or ending at the station.
- We also assume a constant travel time of 1 min for the trains to reach and exit the platforms. For convenience, we added the travel time to the stoppage time of the train as in our simulation once an arrival and departure path has been allotted to a train, that path that path becomes inaccessible for other trains until that train leaves the station. So the net time for which the train occupies that path will include the stoppage and the travel time.
- We also assume equal capacity of all platforms in the station as we did not have the platform capacity data for the different platforms. This assumption can be incorporated very easily in the algorithm by adding the capacity of the platforms in the path choosing functions of the agents.

The box and whisker plot visualise of the performance of the algorithms across the 40 different perturbed schedules of the original platform for given type of perturbation. The orange line in the box shows the median of the data and the top and bottom lines of the box indicate the first (25<sup>th</sup> percentile) and third quartile (75<sup>th</sup> percentile) of the delay data. The upper and lower bars of the whiskers indicate the (Q3+1.5IQR) and (Q1-1.5IQR) range of the data where IQR is the inter-quartile

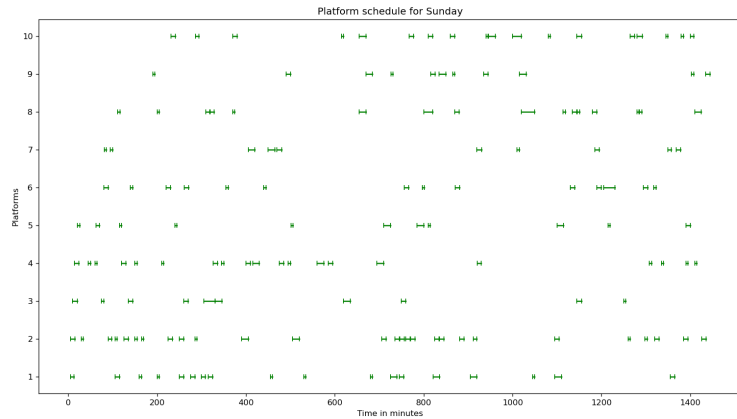


range (Q3-Q1) of the data. The circles indicate the outliers of the data. We present a box and whisker plot to showcase the performance of the algorithms across a variety of perturbations. This plot visualises important statistical properties of the data. A bar chart can also be used to visualize the performance however for 5 different algorithms the graph becomes incomprehensible and hence is not able to provide proper intuition about the delay data for the algorithms.

In the tables mentioned below, the row **# Best** and **# Worst** refer to the number of times the algorithm performed the best and worst respectively in each experiment. For two algorithms, achieving the same delays we give the same rank to both of them. Thus, multiple algorithms can have best or worst performance for a given data point, so the sum of the values in the **# Best** or **# Worst** row may not always sum to 40.

### 7.1.2 Sunday

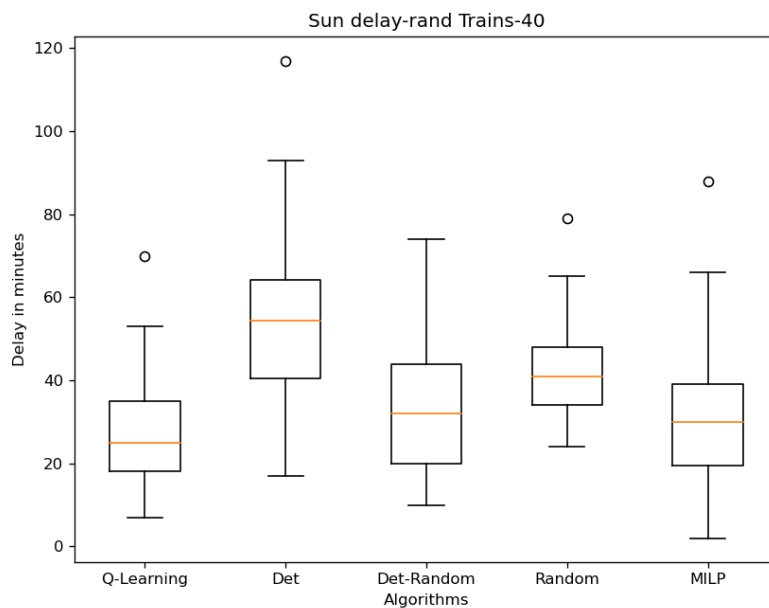
We take a real time-table consisting of 155 trains scheduled for arrival on Sunday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.1, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and discuss the effectiveness of rescheduling trains to minimize the net delay.



**Figure 7.1:** Platform schedule for Sunday

### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.2:** Comparison of Algorithms on small Sunday schedule with a random delay of  $[10,55]$  minutes to all the 40 trains in the schedule.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	7	17	15	24	2
Max Delay	70	117	74	79	88
Median Delay	25	54	32	41	30
# Best	19	0	8	0	18
# Worst	1	27	1	7	5

**Table 7.1:** Statistics of the algorithms on small instance of Sunday Schedule

From Fig 7.2 and Table 7.1, we can see that for the majority of the cases the Q-learning algorithm was able to produce a schedule with the minimum delay. In some cases, it was even able to provide better schedules than the MILP algorithm. It is also interesting to note the performance of the Det-Random algorithm which

was able to provide a good solutions with minimum computational overhead. Another important observation from the data is that the random scheduling algorithm also achieved better performance than the Deterministic platform scheduling algorithm. This goes to show that in case of major delays, it is always better to follow a rescheduling policy rather than obeying the original schedule. The worst performing Deterministic algorithm reached led to a max delay of 117 minutes or almost two hours in this schedule. All other rescheduling algorithms save almost 40 minutes than the Deterministic algorithm. For the worst case where the Det Algorithm generated 117 minutes, the best performing Q-Learning algorithm was able to provide a schedule that only produced a delay of 53 minutes.

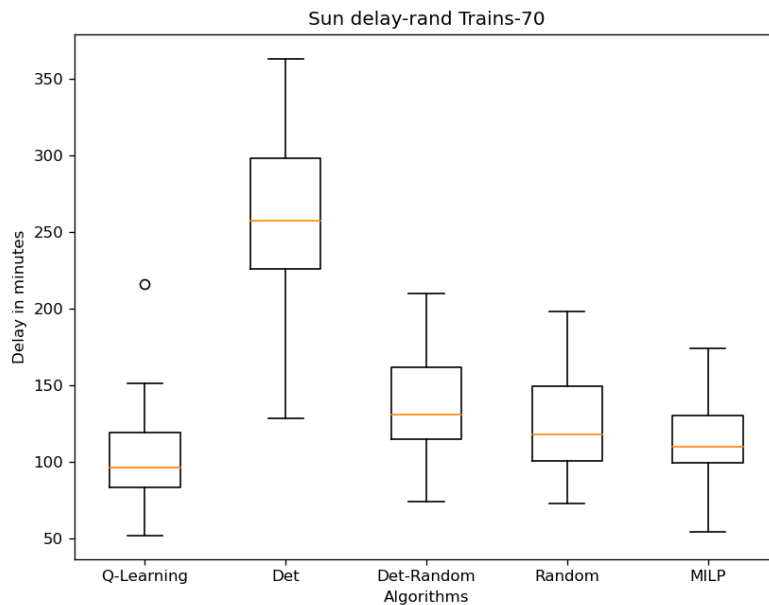
## 24 hour instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from [10,55] minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	52	128	74	73	54
Max Delay	216	363	210	198	174
Median Delay	96	257	131	118	110
# Best	33	0	1	3	8
# Worst	0	40	0	0	0

**Table 7.2:** Statistics of the algorithms on 24 hour instance of Sunday Schedule

From Fig 7.3 and Table 7.2, we can see that for the majority of the cases the Q-learning algorithm was able to produce a schedule with the minimum delay. In this instance the effect of delays is more pronounced and we see the worst performance by the Det algorithm. In many cases, Q-learning was even able to provide better schedules than the MILP algorithm. It is also interesting to note the performance of the Det-Random algorithm which was able to provide a good solutions with minimum computational overhead. Another important observation from the



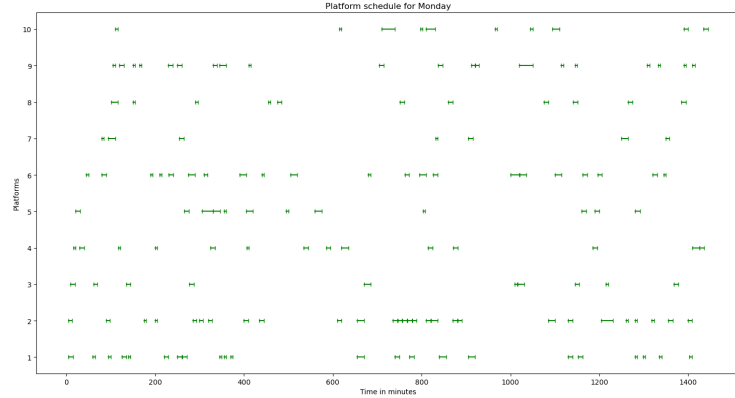
**Figure 7.3:** Comparison of Algorithms on 24 hour Sunday schedule with a random delay of  $[10,55]$  minutes to 70 random trains in the schedule.

data is that the random scheduling algorithm also achieved better performance than the Deterministic platform scheduling algorithm as well as the Det Random algorithm. This goes to show that in case of major delays for a long period, the optimal scheduling strategy deviates a lot from the original schedule. The worst performing Det algorithm led to a max delay of 363 minutes or almost six hours in this schedule. All other rescheduling algorithms save almost 150 minutes than the Deterministic algorithm when considering the max delay. For the worst case where the Det Algorithm generated 363 minutes, the best performing Q-Learning algorithm was able to provide a schedule that only produced a delay of 147 minutes which is almost equal to the delays produced by the MILP schedule of 148 minutes.

### 7.1.3 Monday

We take a real time-table consisting of 155 trains scheduled for arrival on Monday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.4, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and

discuss the effectiveness of rescheduling trains to minimize the net delay.



**Figure 7.4:** Platform schedule for Monday

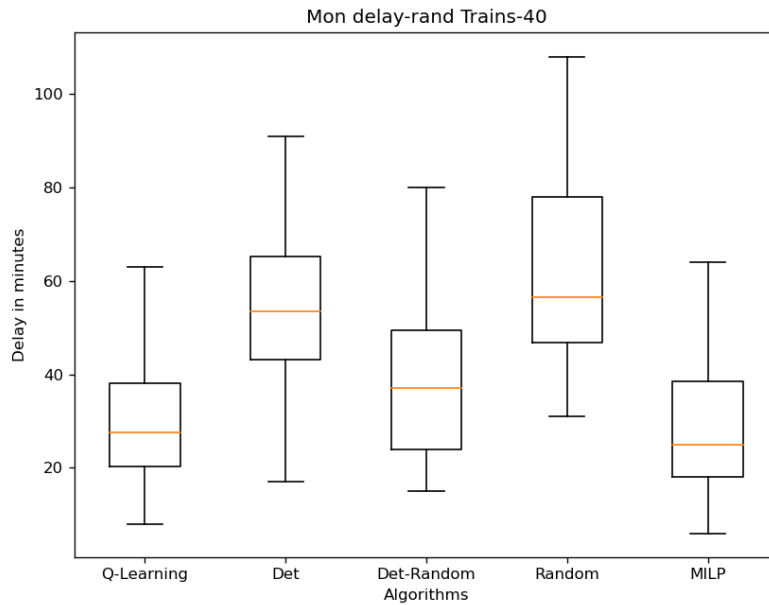
### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	8	17	15	31	6
Max Delay	63	91	80	108	64
Median Delay	27	53	37	56	25
# Best	19	2	1	0	22
# Worst	0	16	2	24	0

**Table 7.3:** Statistics of the algorithms on small instance of Monday Schedule

From Fig 7.5 and Table 7.3, we can see that both the Q-Learning and MILP algorithms performed equally well. It is interesting to note that in this instance, the random algorithm performed worse than Det. For most cases, the MILP algorithm produced the best results however, Q-Learning agent was also close. The Det-Random agent did not perform as well as it did on the Sunday schedules. For the worst case, where the random algorithm generated 108 minutes of delay, MILP



**Figure 7.5:** Comparison of Algorithms on Monday schedule with a random delay of  $[10,55]$  minutes to all the 40 trains in the schedule.

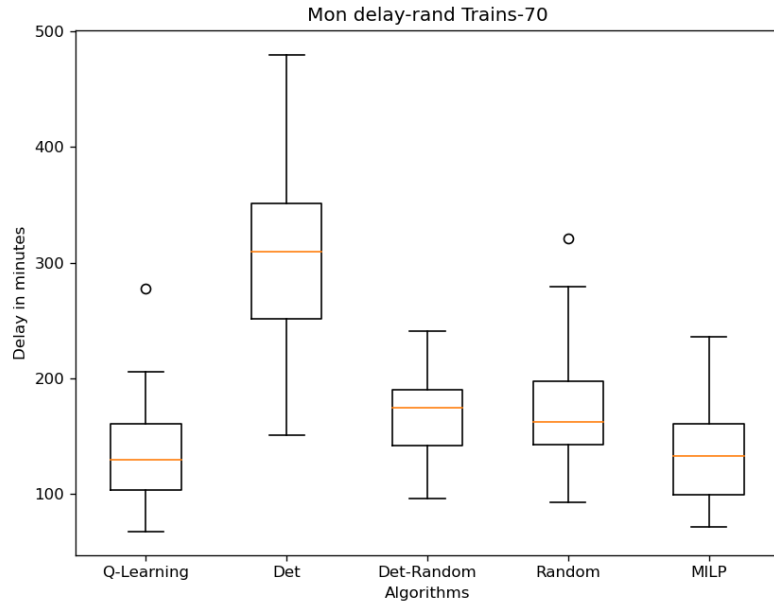
algorithm solved it with a delay of 32 minutes. The solution provided by Q-Learning algorithm generated 48 minutes of delay for this instance. It is quite interesting to see that the Det algorithm performed much better than the random agent for this instance, generating only 77 minutes of delay.

## 24 hour Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	68	151	96	93	72
Max Delay	278	480	241	321	236
Median Delay	129	310	174	162	133
# Best	28	0	6	2	23
# Worst	0	39	1	0	0

**Table 7.4:** Statistics of the algorithms on 24 hour instance of Monday Schedule



**Figure 7.6:** Comparison of Algorithms on 24 hour Monday schedule with a random delay of  $[10,55]$  minutes to 70 random trains in the schedule.

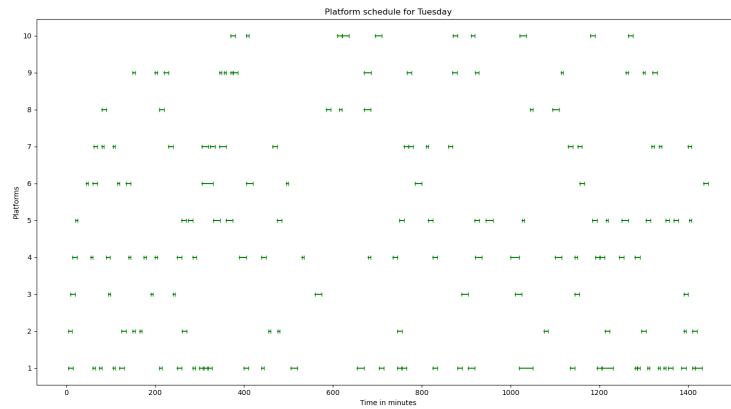
From Fig 7.6 and Table 7.4, we can see that the Q-Learning algorithm performed the best while MILP being a close second. The Det-Algorithm was the worse across all instances. In the worst case, the Det algorithm produced a delay of 480 minutes while Q-learning produced the best schedule with a delay of 162 minutes. Rescheduling trains in this case save almost 5 hours worth of train waiting time.

#### 7.1.4 Tuesday

We take a real time-table consisting of 155 trains scheduled for arrival on Tuesday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.7, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and discuss the effectiveness of rescheduling trains to minimize the net delay.

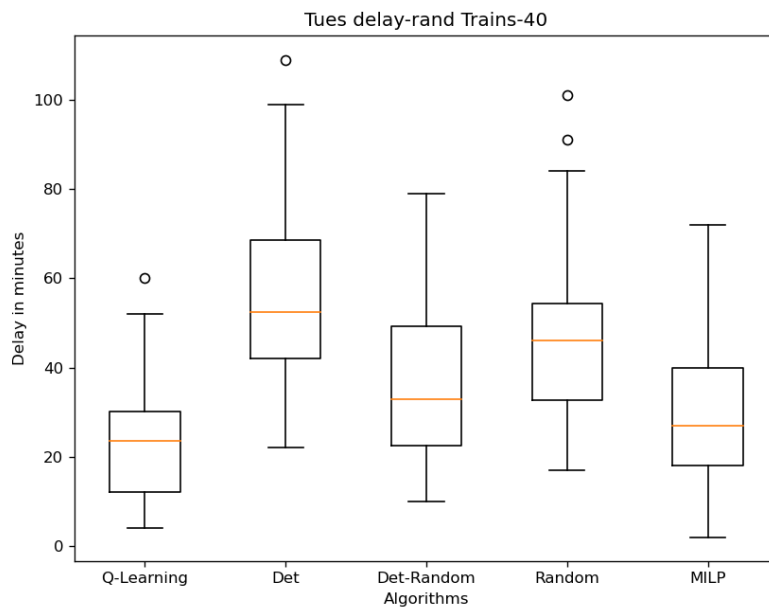
##### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and



**Figure 7.7:** Platform schedule for Tuesday

perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.8:** Comparison of Algorithms on Tuesday schedule with a random delay of  $[10,55]$  minutes to all the 40 trains in the schedule.

From Fig 7.8 and Table 7.5, we can see that the Q-Learning algorithm performed the best. The Det and random algorithm were equally worse across all instances. In the worst case, the Det algorithm produced a delay of 109 minutes while Q-learning produced the best schedule with a delay of 31 minutes. The MILP algorithm did



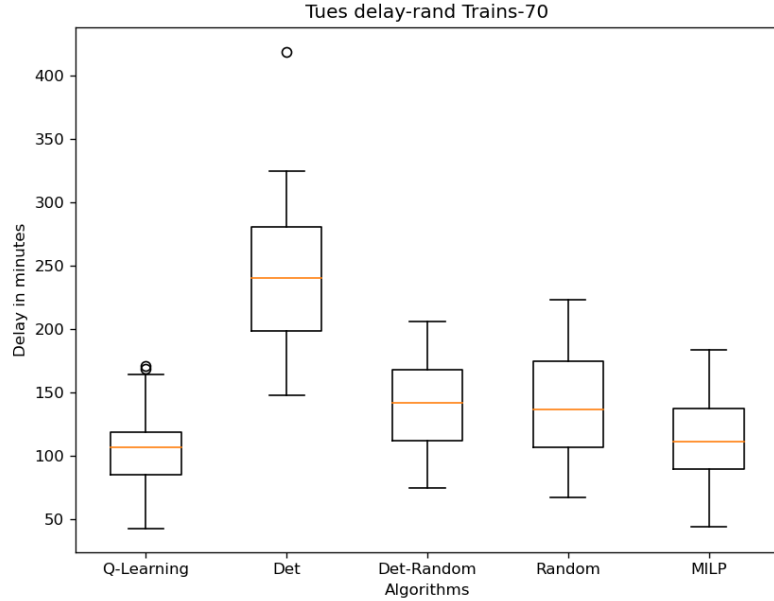
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	4	22	10	17	2
Max Delay	60	109	79	101	72
Median Delay	23	52	33	46	27
# Best	23	0	5	0	14
# Worst	0	26	2	12	1

**Table 7.5:** Statistics of the algorithms on small instance of Tuesday Schedule

not perform as well in this instance as it did in the previous experiments.

## 24 hour Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.9:** Comparison of Algorithms on Tuesday schedule with a random delay of  $[10,55]$  minutes to 70 random trains in the schedule.

From Fig 7.9 and Table 7.6, we can see that the Q-Learning algorithm performed the best. The Det algorithm was the worse across all instances. It is interesting to see that the Det-Random and Random algorithms performed roughly the same. In

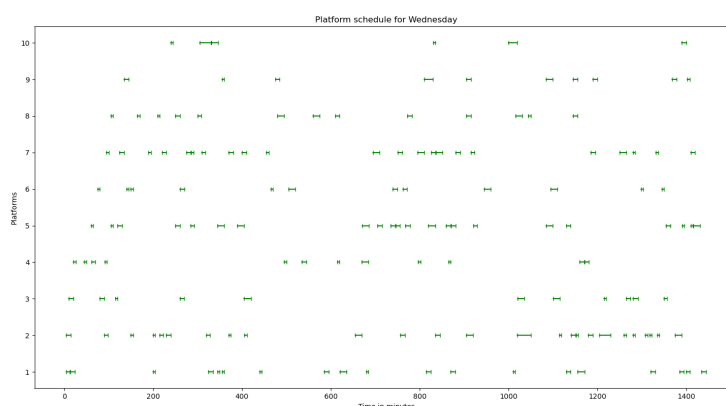
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	43	148	75	67	44
Max Delay	171	419	206	223	184
Median Delay	106	240	142	137	111
# Best	32	0	1	2	14
# Worst	0	39	0	1	0

**Table 7.6:** Statistics of the algorithms on 24 hour instance of Tuesday Schedule

the worst case, the Det algorithm produced a delay of 419 minutes while Q-learning produced the best schedule with a delay of 169 minutes. The MILP algorithm did not perform as well in this instance as it did in the previous experiments.

### 7.1.5 Wednesday

We take a real time-table consisting of 152 trains scheduled for arrival on Wednesday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.10, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and discuss the effectiveness of rescheduling trains to minimize the net delay.

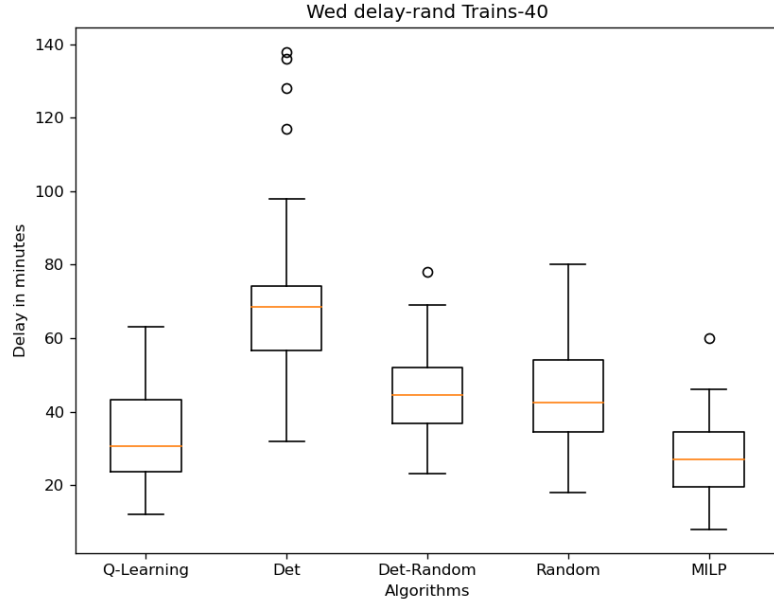


**Figure 7.10:** Platform schedule for Wednesday

#### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and

perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.11:** Comparison of Algorithms on Wednesday schedule with a random delay of  $[10,55]$  minutes to all the 40 trains in the schedule.

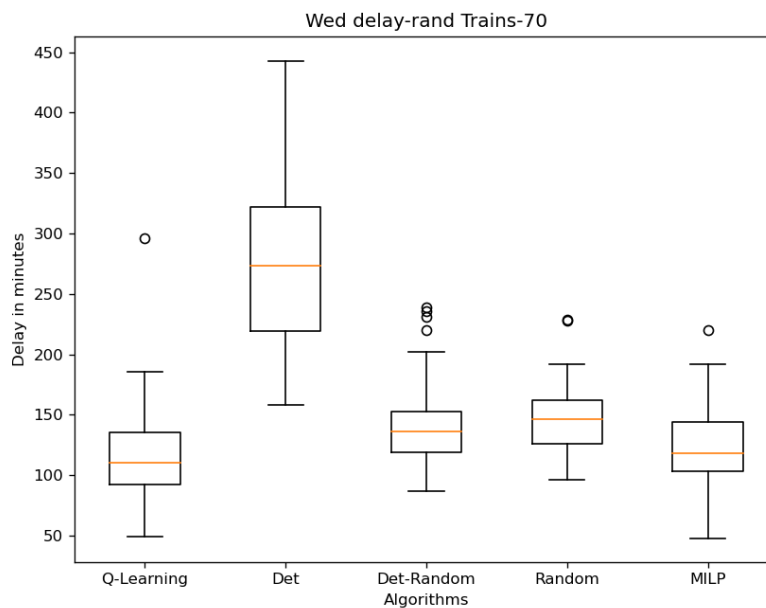
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	12	32	23	18	8
Max Delay	63	138	78	80	60
Median Delay	30	68	44	42	27
# Best	14	0	2	2	25
# Worst	0	35	3	3	0

**Table 7.7:** Statistics of the algorithms on small instance of Wednesday Schedule

From Fig 7.11 and Table 7.7, we can see that the MILP algorithm performed the best. The Det algorithm was the worst across all instances. In the worst case, the Det algorithm produced a delay of 138 minutes while Q-learning algorithm produced the best schedule with a delay of 34 minutes while the MILP solution had a delay of 42. In most of the instances, the MILP algorithm performed the best however, Q-Learning solution was close in almost 14 instances out of the 40.

## 24 hour Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.12:** Comparison of Algorithms on Wednesday schedule with a random delay of  $[10,55]$  minutes to 70 random trains in the schedule.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	49	158	87	96	48
Max Delay	296	443	239	229	220
Median Delay	110	273	136	146	118
# Best	29	0	3	1	15
# Worst	0	40	0	0	0

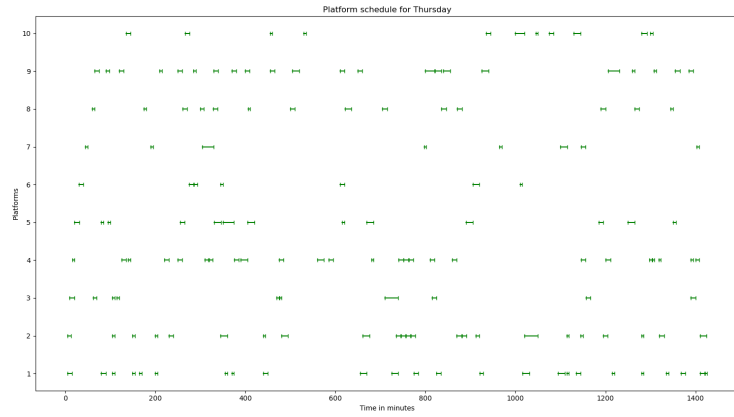
**Table 7.8:** Statistics of the algorithms on 24 hour instance of Wednesday Schedule

From Fig 7.12 and Table 7.8, we can see that the Q-Learning algorithm performed the best with MILP being a close second. The Det algorithm was the worse across all instances. The Det-random algorithm has many outliers and the IQR is also very low. Thus, final solutions found by rescheduling by the Det-Random algorithm were

not as different as it seems from the IQR of the Det algorithm. This algorithm also worked very well across many instances although it was not able to generate the best schedule, in many cases its solution was close to the best solution. In the worst case, the Det algorithm produced a delay of 443 minutes while Q-learning produced the schedule with a delay of 296 minutes. The best schedule in this case was generated by the Det-Random and MILP algorithms. This instance again indicates that the Det-Random is a cheap and quite effective heuristic for scheduling trains.

### 7.1.6 Thursday

We take a real time-table consisting of 157 trains scheduled for arrival on Thursday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.13, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and discuss the effectiveness of rescheduling trains to minimize the net delay.

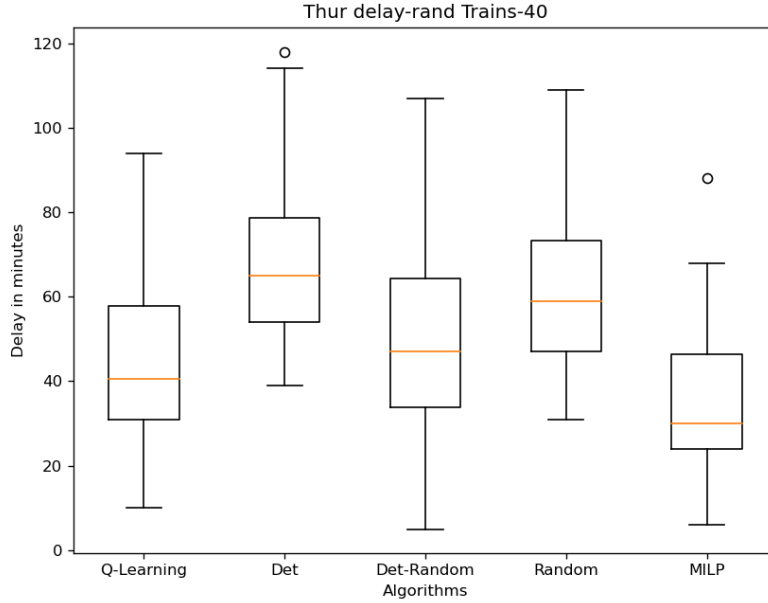


**Figure 7.13:** Platform schedule for Thursday

#### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate

40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.14:** Comparison of Algorithms on Thursday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule.

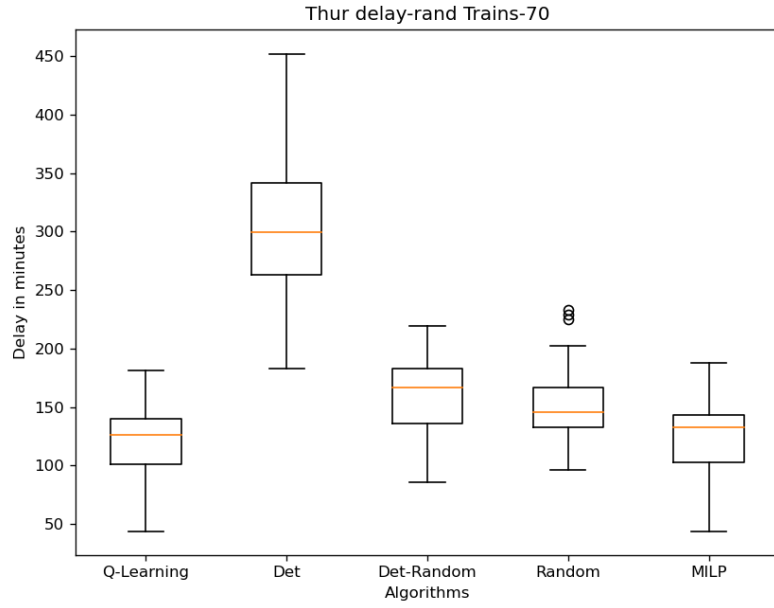
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	10	39	5	31	6
Max Delay	94	118	107	109	88
Median Delay	40	65	47	59	30
# Best	10	0	6	1	27
# Worst	3	21	0	14	3

**Table 7.9:** Statistics of the algorithms on small instance of Thursday Schedule

From Fig 7.14 and Table 7.9, we can see that the MILP algorithm performed the best. The Det and random algorithm were both worst across all instances with random being slightly better. The Det-Random and Q-Learning agents showed similar performance in this instance. In the worst case, the Det algorithm produced a delay of 118 minutes while MILP algorithm produced the best schedule with a delay of 28 minutes, Q-Learning solution was close to the best solution with a delay of 31 minutes. Even the Det-Random algorithm was much better than the worst, with a delay of 38 minutes.

### 24 hour Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from [10,55] minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.15:** Comparison of Algorithms on Thursday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	44	183	86	96	44
Max Delay	181	452	219	233	188
Median Delay	126	299	167	146	133
# Best	28	0	0	3	19
# Worst	0	40	0	0	0

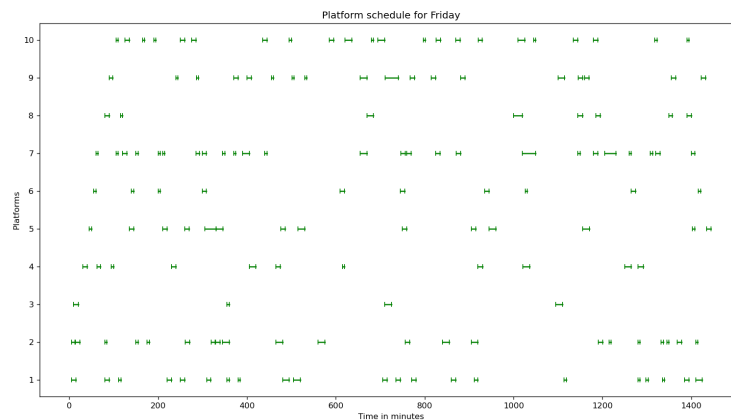
**Table 7.10:** Statistics of the algorithms on 24 hour instance of Thursday Schedule

From Fig 7.15 and Table 7.10, we can see that the Q-Learning algorithm performed the best with MILP being a very close second. The Det algorithm was the worst across all instances. In the worst case, the Det algorithm produced a delay of 452 minutes while MILP produced the best schedule with a delay of 100 minutes

with the Q-learning solution having a delay of 103 minutes. Both of these algorithms have a large IQR in this instance, indicating that the final schedules varied across all the instances. An interesting thing to notice here is that the random agent performed better than the Det-Random agent.

### 7.1.7 Friday

We take a real time-table consisting of 154 trains scheduled for arrival on Friday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.16, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and discuss the effectiveness of rescheduling trains to minimize the net delay.



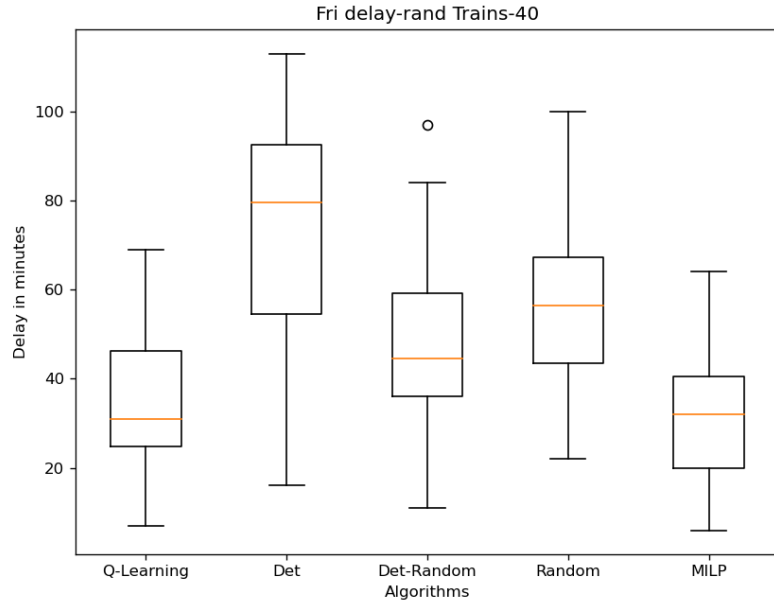
**Figure 7.16:** Platform schedule for Friday

#### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.

From Fig 7.17 and Table 7.11, we can see that the Q-Learning and MILP performed equally well. The Det and random algorithms were the worst across all





**Figure 7.17:** Comparison of Algorithms on Friday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule.

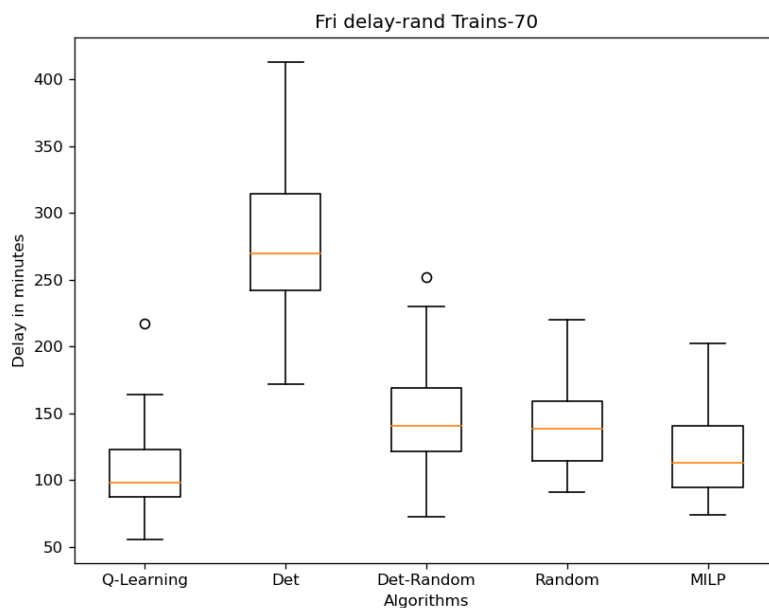
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	7	16	11	22	6
Max Delay	69	113	97	100	64
Median Delay	31	79	44	56	32
# Best	16	0	5	0	20
# Worst	1	27	2	10	1

**Table 7.11:** Statistics of the algorithms on small instance of Friday Schedule

instances. In the worst case, the Det algorithm produced a delay of 113 minutes while Q-learning produced the best schedule with a delay of 45 minutes with the MILP solution producing a delay of 56 minutes.

## 24 hour Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from [10,55] minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.18:** Comparison of Algorithms on Friday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule.

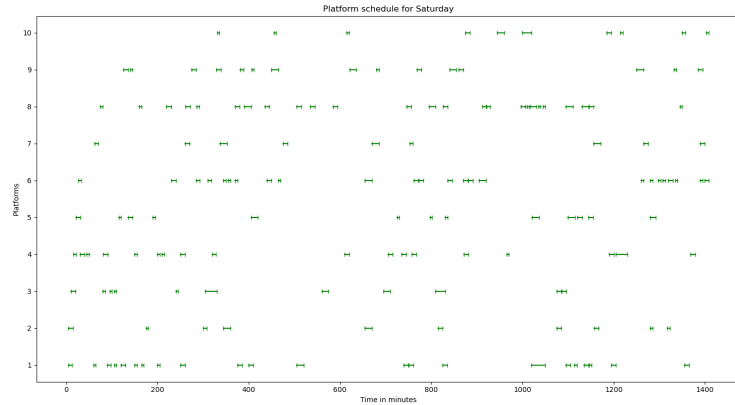
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	56	172	73	91	74
Max Delay	217	413	252	220	202
Median Delay	98	269	140	138	113
# Best	29	0	2	1	15
# Worst	0	40	0	0	0

**Table 7.12:** Statistics of the algorithms on 24 hour instance of Friday Schedule

From Fig 7.18 and Table 7.12, we can see that the Q-Learning algorithm performed the best. The Det algorithm was the worst across all instances. In the worst case, the Det algorithm produced a delay of 413 minutes while Q-learning and MILP both produced the best schedule with a delay of 151 minutes.

### 7.1.8 Saturday

We take a real time-table consisting of 157 trains scheduled for arrival on Saturday. The stoppage times for the trains range from 5 minutes to 30 minutes. In the Fig 7.19, we show a platform schedule that leads to zero platform delays if all the trains arrive on time. The experiments mentioned in this section perturb this schedule and discuss the effectiveness of rescheduling trains to minimize the net delay.

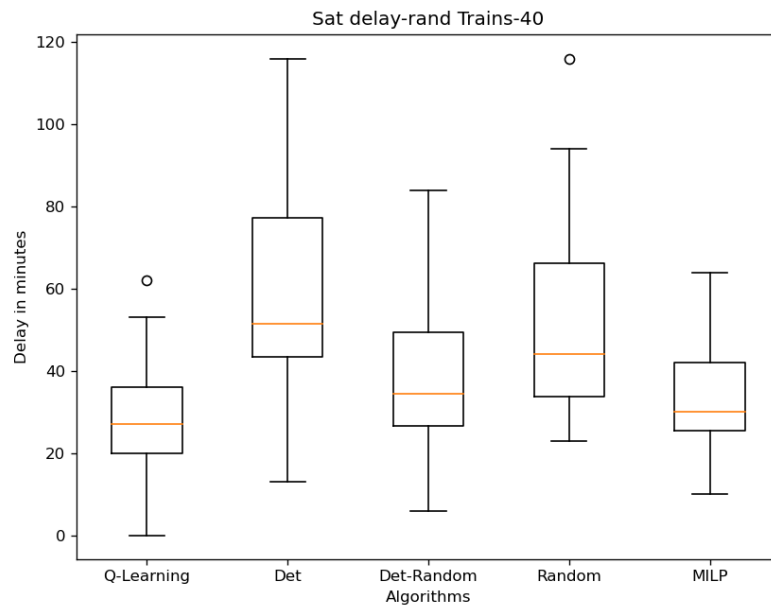


**Figure 7.19:** Platform schedule for Saturday

#### Small Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from  $[10,55]$  minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.

From Fig 7.20 and Table 7.13, we can see that the Q-Learning algorithm performed the best with the MILP algorithm being a close second. The Det and random



**Figure 7.20:** Comparison of Algorithms on Saturday schedule with a random delay of [10,55] minutes to all the 40 trains in the schedule.

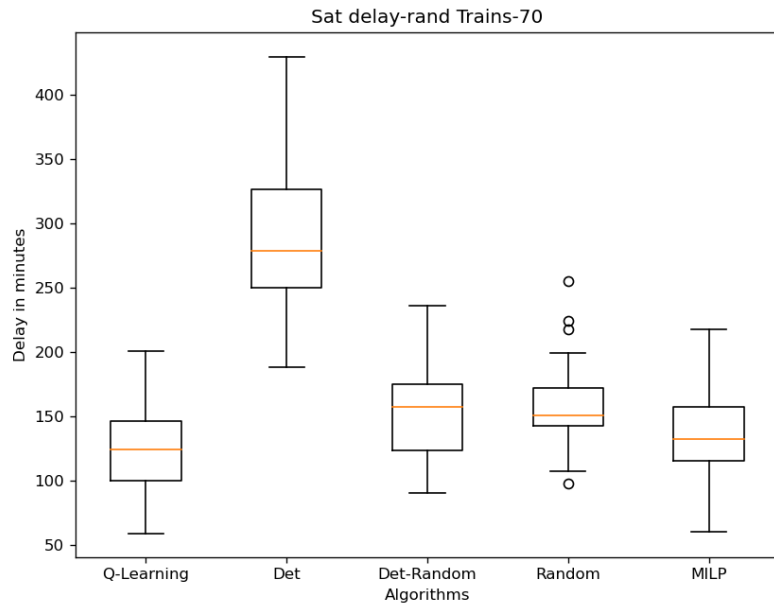
	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	0	13	6	23	10
Max Delay	62	116	84	116	64
Median Delay	27	52	34	44	30
# Best	20	0	5	1	15
# Worst	0	22	1	11	6

**Table 7.13:** Statistics of the algorithms on small instance of Saturday Schedule

algorithm were both worse across all instances. However, Det-Random was quite close to the best solutions in many cases. In the worst case, the Det algorithm produced a delay of 116 minutes while Q-learning produced the best schedule with a delay of 53 minutes with MILP producing a delay of 56 minutes.

## 24 hour Instance of the schedule

In this experiment we take a smaller instance of the original schedule, which consists of 40 trains from roughly 05 : 00 am to 01 : 00 pm. We take the original schedule and perturb all 40 trains by a random delay ranging from [10,55] minutes. We generate 40 different perturbed schedules and observe the performance of all the algorithms on these perturbed schedules.



**Figure 7.21:** Comparison of Algorithms on Saturday schedule with a random delay of [10,55] minutes to 70 random trains in the schedule.

	Q-Learning	Det	Det-Random	Random	MILP
Min Delay	59	188	90	98	60
Max Delay	201	430	236	255	218
Median Delay	124	279	157	150	132
# Best	31	0	3	1	8
# Worst	0	40	0	0	0

**Table 7.14:** Statistics of the algorithms on 24 hour instance of Saturday Schedule

From Fig 7.21 and Table 7.13, we can see that the Q-Learning algorithm performed the best. The Det algorithm was the worst across all instances. In the worst case, the Det algorithm produced a delay of 430 minutes while Q-learning produced the best schedule with a delay of 178 minutes. The MILP algorithm did not perform as well in this instance as it did in the previous experiments. Both Det-Random and random algorithms performed almost the same on this experiment, however, the IQR for random was much less than the other algorithms.

## 7.2 Discussion

Throughout all the experiments, one thing that was most noticeable was that adhering to the original platform schedule in the case of delays is the worst strategy. In many cases, it led to hours of delay compared to other strategies. All the experiments show, that delays lead to conflicts between trains and the original schedule no longer works as the best strategy. One of the main observations that we gained through the main experiments is that even the Det-Random strategy works well for a lot of cases. It does not guarantee the best results all the time, but for a majority of cases it leads to good solutions. This strategy is both computationally inexpensive and can be readily applied without much infrastructure. The data for the Det-Random and random algorithms has been calculated by performing 100 iterations on each data point and then selecting the maximum delay that was generated. The maximum has been taken to account for the worst case that would happen while using these randomized algorithms.

We also saw that MILP based strategies work really well in most cases. Our MILP formulation for the problem has a simple linear cost function. But due to this it sometimes also leads to bad schedules. Other stronger formulations may work better than our current formulation. One of the main disadvantages of the MILP based solutions is the time taken by the solver to generate a feasible result. We observed that this algorithm took almost 20 to 30 minutes to provide a solution for the 24 hour schedules. The solving time is divided among the pattern incompatibility

creation time and the time taken by the solver. The pre-processing time is roughly in the range of 12 to 15 minutes depending upon the number of the trains in the schedule. This is due to the large state-space that the algorithm must create before it can try looking for a solution. In general for the 24 hour schedules, the number of nodes lie in the range of 13000 to 15000. Increasing the granularity, of the delays may lead to better solutions but it also leads to larger number of nodes. In order to complete the Pattern Incompatibility graph, we need to check all pairs of nodes for incompatibility constraints and assign a node appropriately. This leads to checks with a time complexity of  $O(N^2)$ . This check can however be optimized using parallel programming techniques as these checks do not lead to large loop-data dependence. Furthermore, (Caprara et al. 2007a) suggests a better checking algorithm where we only compare nodes that are temporally close to each other. However, there still remains the solver time which varies with the complexity of the problem at hand.

The major advantage of using simulator based techniques is that they are extremely fast. An episode of the simulator for the 24 hour schedule roughly takes 0.02 seconds. Training the Q-learning algorithm for even 2500 epochs leads to a search time of 50 seconds. The Q-learning algorithm was able to come up with the best solutions in the majority of the cases for the 24 hour schedule. The performance was not as good for the smaller instance. One major problem with the Q-learning algorithm is its initialization and the first few steps states that it visits. These initial states greatly influence the final policy that the algorithm converges on. However, we trained the algorithm multiple times on a given 24 hour schedules and only very few times it ended up not converging to the best policy. The solutions generated by the algorithm change in the multiple runs of the algorithm, but in the majority of the cases the solution does not deviate a lot.

An advantage of using simulator based methods is they can be directly applied on the railway station and online algorithms can be developed that take the incoming data and create a solution. These methods also provide a good intuition about what the algorithm is doing and can also be used to create better heuristics for the

scheduling problem. Another use case for the simulator based algorithms is that they can also be used to create new schedules without any knowledge of existing platform schedules. The final schedule that is generated is free from conflicts arriving on the station and those can be used as starting schedules for experts to plan and create better scheduling algorithms. The simulator based algorithms can also be used to simulate rare events happening on the station and can provide experts with more knowledge about the problems that they might face at real stations.



# Chapter 8

## Conclusions

In this thesis, we investigate the propagation of delays in a railway schedule due to the lack of unavailability of platforms and entry/exit lines in the station. We have proposed two methods to solve the train platforming problem. We also present a station simulator to simulate events at the station. We have performed multiple experiments on the Kanpur Central Station to observe delay propagation due to platform unavailability. Through these experiments we observe that non-rescheduling trains at the station leads to delays of over 6 to 8 hours. In our experiments, we considered schedules for a single day. However, in reality these delays spillover to the next day causing more delays in the schedule. We also observed that even simple rescheduling strategies like Det-Random, work quite well to minimize the delay in most cases. They do not provide the best solution but are extremely easy to apply in reality while requiring minimum infrastructure and computational resources.

Through our experiments, we show that our proposed Q-learning based scheduling agent generates the best schedule compared to the other mentioned algorithms in most cases. The MILP algorithm also works quite well for many cases with a very simple linear objective. One of the main advantages that the Q-learning algorithm has over the MILP algorithms is its computation time. However, the Q-learning agent does not work as well while dealing with schedules with very small delays. In those cases, the Det and Det-Random algorithms achieve good solutions as they build up on the original zero delay schedule. Thus, our Q-learning agent is most

useful in cases where there are very tight schedules and there are major disruptions to the trains.

## 8.1 Scope for further work

In our study, we solve the problem in a more simplified state than it actually is in reality. We observe the problem only from the perspective of the station and its delays. However, there are many other variables that we have not considered in this study. Train rescheduling is important, however, it also leads to large uncertainties for the passengers on the platforms as they need to shift to the different platforms after the rescheduling. It is quite easy to overcome this problem to some extent in our approach, by giving passing the trains a bit early to the agent and keeping track of the future state of the station.

Another assumption that we took was that all trains have the same occupational capacity. This is, however, not true in real life and one needs to take this into account while scheduling the trains. We had to take this assumption as we did not have the data for the platform capacity. This assumption can also be accounted quite easily by incorporating the capacity information in the *possible\_arrival\_paths* and *possible\_departure\_paths* functions of the agent to supply the paths to those platforms that satisfy the capacity of the requesting train. Another modification that needs to be done for real life application of our approach, is the position of the pit and washing lines in the simulator. These need to be added in the station topology to account for the proper maintenance of those trains that either stop or start at the station.

One major change that needs to be done in our simulation algorithm, is the usage of partial train allocations. We currently consider locking paths when a train enters till the time it leaves the station. This, however, is not an optimal strategy as for a train that has a stoppage time of let's say 30 minutes, the incompatible paths would be locked for the whole 30 minutes. This is wasted time as once the train has stopped at the platform, any other train with a shorter stoppage time would not get

access to the locked paths for the whole of 30 minutes. This can be incorporated into the simulator by slightly changing the step function of the environment and the act function of the agent to first only schedule an arrival path and then wait until the departure time to schedule a departure path. This would lead to a another type of delay in the station, which would be related to the extra time the train spends on the platform. We tried the current approach as we were trying to mimic the approach used by (Cacchiani et al. 2015) while developing the pattern incompatibility graph.

In our current formulation, we do not take into account the speed and acceleration of the train within the station. In a realistic setting, these have to be taken into account as well. We used a constant travel time, to simplify our problem formulation. One justification for this is, that inside the station the trains do not need to travel at high speeds and thus almost all trains would travel at similar speeds and would thus take similar amounts of time for their arrival and departure.

As part of a future work, we need to reformulate the state description for our problem to more readily take into account the future trains and the station topology. One such way is to use LSTMs or some other memory based model to account for the future trains arriving in the station while deciding upon the arrival and departure paths for a train. Recently, graph neural networks have also been used for reinforcement learning based scheduling algorithms. These networks work on message passing and are able capture the topology of the station very well. We also need to formulate a method to account for delays and how much these delays deviate the original schedule. This would lead to some standard policies that can be used in the case of delays without the need for recalculation of the perturbed schedule every time.



# References

- Sahai, Vivek, Ameya Pimpalkhare, and Paresh Rawal (2016). *Saral Timetable for the Western Railway Suburban System: An Angioplasty of Mumbai's Lifeline*. Mumbai, Maharashtra, India: Observer Research Foundation Mumbai.
- Comptroller General, India (2018). *Report on Augmentation of Station Line Capacity on selected stations in Indian Railways for the year ended March 2017*. Union Government (Railways).
- Cordeau, Jean-Francois, Paolo Toth, and Vigos Daniele (1998). "A Survey of Optimization Models for Train Routing and Scheduling". In: *Transportation Science* 32.4, pp. 380–404. ISSN: 00411655, 15265447. URL: <http://www.jstor.org/stable/25768836> (visited on 06/11/2022).
- Lusby, Richard et al. (Oct. 2011). "Railway track allocation: Models and methods". In: *OR Spectrum* 33, pp. 843–883. DOI: [10.1007/s00291-009-0189-0](https://doi.org/10.1007/s00291-009-0189-0).
- Carey, Malachy and Sinead Carville (2003). "Scheduling and platforming trains at busy complex stations". In: *Transportation Research Part A: Policy and Practice* 37.3, pp. 195–224. ISSN: 0965-8564. DOI: [https://doi.org/10.1016/S0965-8564\(02\)00012-5](https://doi.org/10.1016/S0965-8564(02)00012-5). URL: <https://www.sciencedirect.com/science/article/pii/S0965856402000125>.
- Cardillo, Dorotea De Luca and Nicola Mione (Apr. 1998). "k L-list [ $\lambda$ ] colouring of graphs". In: *European Journal of Operational Research* 106.1, pp. 160–164. URL: <https://ideas.repec.org/a/eee/ejores/v106y1998i1p160-164.html>.
- Billionnet, Alain (2003). "Using Integer Programming to Solve the Train Platforming Problem". In: *Transportation Science* 37, pp. 213–222. URL: <https://hal.archives-ouvertes.fr/hal-01124706>.
- Ghoseiri, Keivan, Ferenc Szidarovszky, and Mohammad Asgharpour (Feb. 2004). "A Multi - Objective Train Scheduling Model and Solution". In: *Transportation Research Part B: Methodological* 38, pp. 927–952. DOI: [10.1016/j.trb.2004.02.004](https://doi.org/10.1016/j.trb.2004.02.004).
- Zwaneveld, Peter J., Leo G. Kroon, and Stan P. M. van Hoesel (Jan. 2001). "Routing trains through a railway station based on a node packing model". In: *European Journal of Operational Research* 128.1, pp. 14–33. URL: <https://ideas.repec.org/a/eee/ejores/v128y2001i1p14-33.html>.
- Chakroborty, Partha and Durgesh Vikram (Feb. 2008). "Optimum assignment of trains to platforms under partial schedule compliance". In: *Transportation Research Part B: Methodological* 42.2, pp. 169–184. URL: <https://ideas.repec.org/a/eee/transb/v42y2008i2p169-184.html>.
- Sels, Peter et al. (Mar. 2014). "The train platforming problem: The infrastructure management company perspective". In: *Transportation Research Part B: Methodological* 61, pp. 55–72. DOI: [10.1016/j.trb.2014.01.004](https://doi.org/10.1016/j.trb.2014.01.004).

- Bai, L. et al. (Jan. 2014). “A mixed-integer linear program for routing and scheduling trains through a railway station”. In: *ICORES 2014 - Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems*, pp. 445–452.
- Akyol, Emine (2017). “Train Platforming Problem Solving”. PhD thesis. University of Pardubice.
- Pellegrini, Paola, Grégory Marliere, and Joaquin Rodriguez (Jan. 2014). “Optimal train routing and scheduling for managing traffic perturbations in complex junctions”. In: *Transportation Research Part B: Methodological*, p58–80. DOI: [10.1016/j.trb.2013.10.013](https://doi.org/10.1016/j.trb.2013.10.013). URL: <https://hal.archives-ouvertes.fr/hal-00930241>.
- Cacchiani, Valentina, Laura Galli, and Paolo Toth (Jan. 2014). “A tutorial on non-periodic train timetabling and platforming problems”. In: *EURO Journal on Transportation and Logistics* 4. DOI: [10.1007/s13676-014-0046-4](https://doi.org/10.1007/s13676-014-0046-4).
- Caprara, Alberto, Laura Galli, and Paolo Toth (2007a). “04. Solution of the Train Platforming Problem”. In: *7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'07)*. Ed. by Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa. Vol. 7. OpenAccess Series in Informatics (OASICS). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-939897-04-0. DOI: [10.4230/OASICS.ATMOS.2007.1174](https://doi.org/10.4230/OASICS.ATMOS.2007.1174). URL: <http://drops.dagstuhl.de/opus/volltexte/2007/1174>.
- D’Ariano, Andrea, Dario Pacciarelli, and Marco Pranzo (Dec. 2007). “A branch and bound algorithm for scheduling trains in a railway network”. In: *European Journal of Operational Research* 183.2, pp. 643–657. URL: <https://ideas.repec.org/a/eee/ejores/v183y2007i2p643-657.html>.
- Dewilde, Thijs et al. (2013). “Robust Railway Station Planning: an Interaction between Routing, Timetabling and Platforming”. eng. In: *Journal of Rail Transport Planning and Management* 3.3, pp. 68–77. ISSN: 2210-9706. URL: [https://lirias.kuleuven.be/retrieve/279251\\$\\$DRobust%20railway%20station%20planning\\_.pdf%20\[freely%20available\]](https://lirias.kuleuven.be/retrieve/279251$$DRobust%20railway%20station%20planning_.pdf%20[freely%20available]).
- Dessouky, Maged et al. (Feb. 2006). “An exact solution procedure to determine the optimal dispatching times for complex rail networks”. In: *IIE Transactions* 38. DOI: [10.1080/074081791008988](https://doi.org/10.1080/074081791008988).
- Tornquist, J. and J.A. Persson (2005). “Train Traffic Deviation Handling Using Tabu Search and Simulated Annealing”. In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 73a–73a. DOI: [10.1109/HICSS.2005.641](https://doi.org/10.1109/HICSS.2005.641).
- Corman, Francesco, Rob M. P. Goverde, and Andrea D’Ariano (2009). “Rescheduling Dense Train Traffic over Complex Station Interlocking Areas”. In: *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*. Ed. by Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis. Vol. 5868. Lecture Notes in Computer Science. Springer, pp. 369–386. DOI: [10.1007/978-3-642-05465-5\\_16](https://doi.org/10.1007/978-3-642-05465-5_16). URL: [https://doi.org/10.1007/978-3-642-05465-5\\_16](https://doi.org/10.1007/978-3-642-05465-5_16).
- Reinforcement Learning for Train Movement Planning at Railway Stations* (Sept. 2020). Auckland, New Zealand. URL: [https://ala2020.vub.ac.be/papers/ALA2020\\_paper\\_11.pdf](https://ala2020.vub.ac.be/papers/ALA2020_paper_11.pdf).

- Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN: 0262039249.
- Prasad, Rohit, Harshad Khadilkar, and Shivaram Kalyanakrishnan (2021). “Optimising a Real-Time Scheduler for Indian Railway Lines by Policy Search”. In: *2021 Seventh Indian Control Conference (ICC)*, pp. 75–80. DOI: [10.1109/ICC54714.2021.9703176](https://doi.org/10.1109/ICC54714.2021.9703176).
- Khadilkar, Harshad (2019). “A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines”. In: *IEEE Transactions on Intelligent Transportation Systems* 20, pp. 727–736.
- Šemrov, Darja et al. (Apr. 2016). “Reinforcement learning approach for train rescheduling on a single-track railway”. In: *Transportation Research Part B Methodological* 86, pp. 250–267. DOI: [10.1016/j.trb.2016.01.004](https://doi.org/10.1016/j.trb.2016.01.004).
- Ning, Lingbin et al. (2019). “A Deep Reinforcement Learning Approach to High-Speed Train Timetable Rescheduling under Disturbances”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. Auckland, New Zealand: IEEE Press, pp. 3469–3474. DOI: [10.1109/ITSC.2019.8917180](https://doi.org/10.1109/ITSC.2019.8917180). URL: <https://doi.org/10.1109/ITSC.2019.8917180>.
- Riedmiller, Simone C. and Martin A. Riedmiller (1999). “A Neural Reinforcement Learning Approach to Learn Local Dispatching Policies in Production Scheduling”. In: *IJCAI*.
- Runarsson, Thomas, Marc Schoenauer, and Michèle Sebag (Oct. 2012). “Pilot, Roll-out and Monte Carlo Tree Search Methods for Job Shop Scheduling”. In: DOI: [10.1007/978-3-642-34413-8\\_12](https://doi.org/10.1007/978-3-642-34413-8_12).
- Brucker, Peter (2001). *Scheduling Algorithms*. 3rd. Berlin, Heidelberg: Springer-Verlag. ISBN: 3540415106.
- Tesauro, Gerald (Mar. 1995). “Temporal Difference Learning and TD-Gammon”. In: *Commun. ACM* 38.3, pp. 58–68. ISSN: 0001-0782. DOI: [10.1145/203330.203343](https://doi.org/10.1145/203330.203343). URL: <https://doi.org/10.1145/203330.203343>.
- Silver, David et al. (Dec. 2017). “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In.
- Zhang, Cong et al. (2020). “Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning”. In: *ArXiv* abs/2010.12367.
- Nazari, MohammadReza et al. (2018). “Reinforcement Learning for Solving the Vehicle Routing Problem”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: <https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf>.
- Mohanty, Sharada Prasanna et al. (2020). “Flatland-RL : Multi-Agent Reinforcement Learning on Trains”. In: *CoRR* abs/2012.05893. arXiv: [2012.05893](https://arxiv.org/abs/2012.05893). URL: <https://arxiv.org/abs/2012.05893>.
- Cacchiani, Valentina, Laura Galli, and Paolo Toth (2015). “A tutorial on non-periodic train timetabling and platforming problems”. English. In: *EURO Journal on Transportation and Logistics* 4.3. To the memory of our friend and colleague Alberto Caprara., pp. 285–320. DOI: [10.1007/s13676-014-0046-4](https://doi.org/10.1007/s13676-014-0046-4).
- Hirashima, Yoichi (2011). “A Reinforcement Learning Method for Train Marshaling Based on Movements of Locomotive”. In.
- Caprara, Alberto, Laura Galli, and Paolo Toth (2007b). “04. Solution of the Train Platforming Problem”. In: *7th Workshop on Algorithmic Approaches for Trans-*

- portation Modeling, Optimization, and Systems (ATMOS'07)*. Ed. by Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa. Vol. 7. OpenAccess Series in Informatics (OASICS). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-939897-04-0. DOI: [10.4230/OASICS.ATMOS.2007.1174](https://doi.org/10.4230/OASICS.ATMOS.2007.1174). URL: <http://drops.dagstuhl.de/opus/volltexte/2007/1174>.
- Sels, P. et al. (2014). “The train platforming problem: The infrastructure management company perspective”. In: *Transportation Research Part B: Methodological* 61.C, pp. 55–72. DOI: [10.1016/j.trb.2014.01.004](https://doi.org/10.1016/j.trb.2014.01.004). URL: <https://ideas.repec.org/a/eee/transb/v61y2014icp55-72.html>.
- Liu, Shi and Erhan Kozan (2011). “Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model”. In: *Transportation Science* 45.2, pp. 175–198. DOI: [10.1287/trsc.1100.0332](https://doi.org/10.1287/trsc.1100.0332). URL: <https://eprints.qut.edu.au/48664/>.
- Dai, Lu (2018). “A machine learning approach for optimisation in railway planning”. In.
- Zhou, Wenliang, Xiaorong You, and Wenzhuang Fan (2020). “A Mixed Integer Linear Programming Method for Simultaneous Multi-Periodic Train Timetabling and Routing on a High-Speed Rail Network”. In: *Sustainability* 12.3. ISSN: 2071-1050. DOI: [10.3390/su12031131](https://doi.org/10.3390/su12031131). URL: <https://www.mdpi.com/2071-1050/12/3/1131>.
- Garrisi, Gianmarco and Cristina Cervelló-Pastor (2020). “Train-Scheduling Optimization Model for Railway Networks with Multiplatform Stations”. In: *Sustainability* 12.1. ISSN: 2071-1050. DOI: [10.3390/su12010257](https://doi.org/10.3390/su12010257). URL: <https://www.mdpi.com/2071-1050/12/1/257>.
- Agasucci, Valerio, Giorgio Grani, and Leonardo Lamorgese (2020). “Solving the single-track train scheduling problem via Deep Reinforcement Learning”. In: *CoRR* abs/2009.00433. arXiv: [2009.00433](https://arxiv.org/abs/2009.00433). URL: <https://arxiv.org/abs/2009.00433>.
- Nygren, Erik et al. (Oct. 2017). *Reinforcement Learning for Railway Scheduling: Overcoming Data Sparseness through Simulations*. DOI: [10.13140/RG.2.2.22843.11041](https://doi.org/10.13140/RG.2.2.22843.11041).
- Liao, Jinlin et al. (Dec. 2020). “A Real-Time Train Timetable Rescheduling Method Based on Deep Learning for Metro Systems Energy Optimization under Random Disturbances”. In: *Journal of Advanced Transportation* 2020, pp. 1–14. DOI: [10.1155/2020/8882554](https://doi.org/10.1155/2020/8882554).
- Akyol, Emine et al. (July 2017). “A New Scheduling Approach to Train Platforming Problem”. In: *Perner’s Contacts XII*, pp. 5–18.