Utrecht University

Master Thesis

Causal discovery from train network data with background knowledge

Daily supervisor:
Francisco Nunes Ferreira
Quialheiro Simoes

Main supervisor:
Thijs van Ommen

Author:
Vera Schoonderwoerd
(SN: 6142818)

Second supervisor:
Mehdi Dastani

ProRail supervisors:
Emdzad Sehic
Wilco Tielman

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computing Science in the

Faculty of Science
Graduate School of Natural Sciences

29th June, 2023

ProRail

Universiteit Utrecht

Abstract

One delayed train could influence the punctuality of other trains in its area. Currently, the Traffic Controllers of ProRail use their own knowledge to predict the delays of the trains, where they also include the delays of other trains as a factor. ProRail want to research if it is feasible to create a decision support system, where the Traffic Controllers are aided in making predictions about delayed trains and how to intervene to minimize the disruptions. One of the first stepping stones is to create a delay prediction system, and that is what this thesis focuses on. Our goal is the exploration of causal analysis applied to delay data, and this result is included in a delay prediction model. The causal relations between trains are captured in a Structural Causal Model (SCM). Creating the SCM involves two steps: finding the causal graph, and learning the assignment functions. The causal graph is identified by applying background knowledge to the PC-algorithm, which reduces the search space. The assignment functions are learned by training multiple Neural Networks. The result of this thesis is a prediction system that includes causal relations between trains, referred to as *possible train interactions*, as input to predict the delays of the trains at its next time tabling point. The model performs similarly to an existing model and shows potential for improvement in further research.

Acknowledgements

I would like to begin by expressing my sincere gratitude and appreciation to the following people who have helped me during my thesis.

This project would have been impossible without the support of my daily supervisor, Francisco; thank you for all the support and the extensive meetings we had to carry out this research. He guided me by listening to my puzzling thoughts and translating them into more structured ideas. I also want to express my gratitude to my main supervisor, Thijs, for the meetings we had and providing unique ideas to tackle the problems I encountered. I want to thank both of you for being flexible and supportive throughout this research, allowing me to pursue my personal field of interest within this thesis topic. It has been greatly appreciated.

I would also like to thank Wilco, for sharing additional information about the train data and providing insights into the complexity of these processes at a more detailed level. I had no idea that there were so many details to consider when working with trains, and I think my internship was a valuable learning experience. I would like to thank Emdzad for providing me information about ProRail itself, introducing me to colleagues that could help me further with my project, and providing opportunities for me to visit interesting ProRail locations, which have enriched my understanding of the organization. Also, he has provided me first-hand information about the rail infra and train dispatchers; for instance, he arranged a train dispatcher course for one midday, which was a unique experience!

Lastly, I would like to thank my family for the moral support they gave me during my thesis. They were always interested in my research and I could elaborate on my problems when I was stuck. In particular, I would like to thank my partner Tycho for always being there for me. His listening ear and encouragement helped me throughout my thesis, and I am deeply grateful for his support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis provides an overview of the research that is conducted at ProRail. This section starts with an overview of ProRail, providing a brief introduction to the organization. Then we elaborate on what the challenges in Traffic Control are, followed by the problem statement for the use of a decision support system. Subsequently, the context of the project is provided, and lastly, the research questions are outlined.

1.1 ProRail

ProRail is a railway operator that focuses on maintaining and improving the railway network of the Netherlands. They are also responsible for directing the train traffic down the track [ProRail n.d.(a)]. To put it in perspective: every year, there are 160 million kilometers of train travel. There are 398 stations, 11.602 railway signals (traffic lights) and 6260 track switches that need to be maintained. In 2021, 4655 people were working at ProRail to maintain, among other things, these enormous amount of components [ProRail 2021a]. The mission of ProRail is to connect people and businesses by rail, and they make sure that it is safe on and around the track [ProRail 2021b]. This research is conducted at the department *Innovatie en Technologische Vernieuwing* (Innovation and Technical Renewal in English) and focuses on identifying delays caused by train interaction on the railway by means of the Structural Causal Model framework.

1.2 Challenges in Traffic Control

The tracks in the Netherlands are divided in 12 regions, with a Traffic Control Center (TTC) per region [ProRail 2021c], as shown in Figure 1.1. Each TTC is divided in

several areas, and each area is a workplace for a train dispatcher. An example of a TCC and its divided areas is shown in Figure 1.2, where the regions of Eindhoven are divided between different workplaces of the train dispatchers. Train dispatchers within the same TTC sit next to each other, which makes the communication between them fast. The communication between train dispatchers of other TTCs is slower, since they have to call each other or communicate by means of a control system.

FIGURE 1.1: Traffic Control Centers in the Netherlands [NOS 2022]

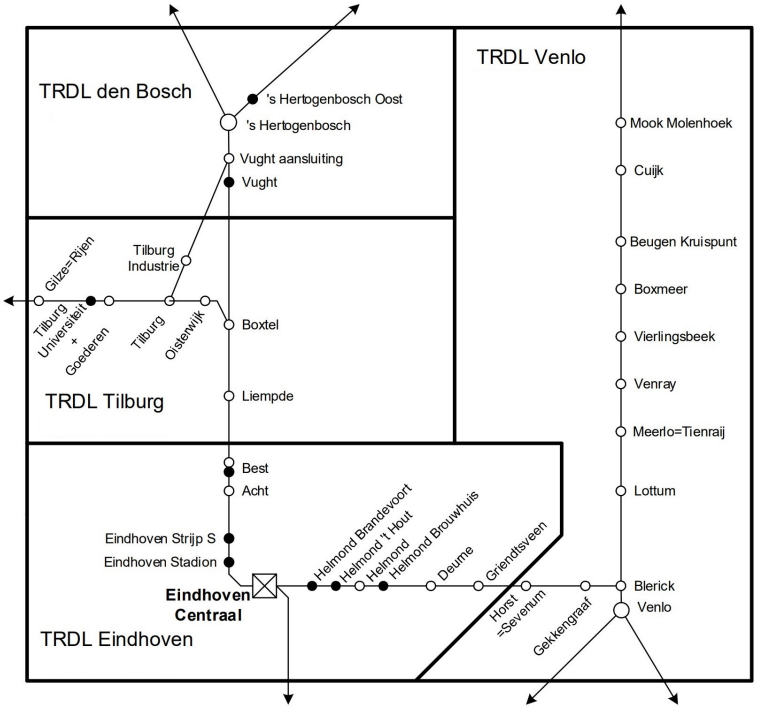

FIGURE 1.2: Decomposition of the work areas within the Traffic Control Center of Eindhoven. The black dots denote a smaller station, the white dots denote a bigger station. The crossed box at Eindhoven Centraal denotes the station where the rail traffic control system is located. [Movares et al. 2022]

The train dispatcher enables the routes of the trains and manages the throughput of trains, by controlling signs and switches. When there are disturbances, they need to solve them by rescheduling or cancelling trains. In situations without disruptions, the control system ARI (*Automatische Rijweg Instelling* or Automatic Routing System) takes over the work of the train dispatcher and enables the routes for each train automatically. ARI works according to given plan lines, which detail the routes that need to be enabled for specific trains. These plans are ordered by the configuration time for the route. Figure 1.3 presents an example of such plan lines, with the first rule highlighted in yellow.

Train number	Activity	Planned time for train	Configuration time for route	From	To
5217	V	7:17	7:16	2	GB
9617	A	7:19	7:16	BA	5
49613	D	7:20	7:17	HB	CA

FIGURE 1.3: Sketch of plan screen [ProLeren n.d.(a)]. The first line is highlighted yellow

ARI works as follows. ARI checks only if the first line of the plan can be enabled. For this, three things need to be confirmed. 1) The train is present at the correct location. 2) The train arrives at the correct time (which may deviate for the trains coming from the open track, there is a predetermined time interval allowed). 3) The route of the plan line is non-conflicting (i.e., all sections that need to be enabled are not in use) [ProLeren n.d.(a)].

However, ARI has limits. Particularly, if it cannot confirm the first plan rule, it waits until it can do so, during which time other traffic must wait. Thus, when the ordering of two trains is changed, a deadlock occurs (since now the second rule comes before the first rule) and ARI will fail to enable the routes in this situation. In such cases, the train dispatcher disables ARI, and enables the routes manually.

A distinction can be made between *disturbance*, which are small incidents, and *disruption*, which are large incidents for which significant changes are necessary [Nielsen et al. 2012]. Within the time schedule, buffers are created, such that small disturbances are resolved automatically [ProRail 2018]. To minimize disruptions, it is sometimes beneficial to change some small elements of the schedule, such as changing an ordering of trains or skipping a station. Such actions are described in the Trein Afhandelings Document (*TAD*, English: Train Handling Document) [ProLeren n.d.(c)]. With the work of Wieringen (2019), the TAD's are implemented in a simulation of a train dispatcher workplace of ProRail by the use of a multi-agent system.

1.3 Problem statement for the implementation of a decision support system

The disruption within an area may propagate to other areas in the country, since trains have many interactions with other trains at other locations. For example, on 3 February 2012 there were many out-of-control situations due to extreme weather conditions. The delay propagation is visualized in Figure 1.4 [Dekker et al. 2021], where the disruption started between Amsterdam and Utrecht and propagated towards the south of the country.

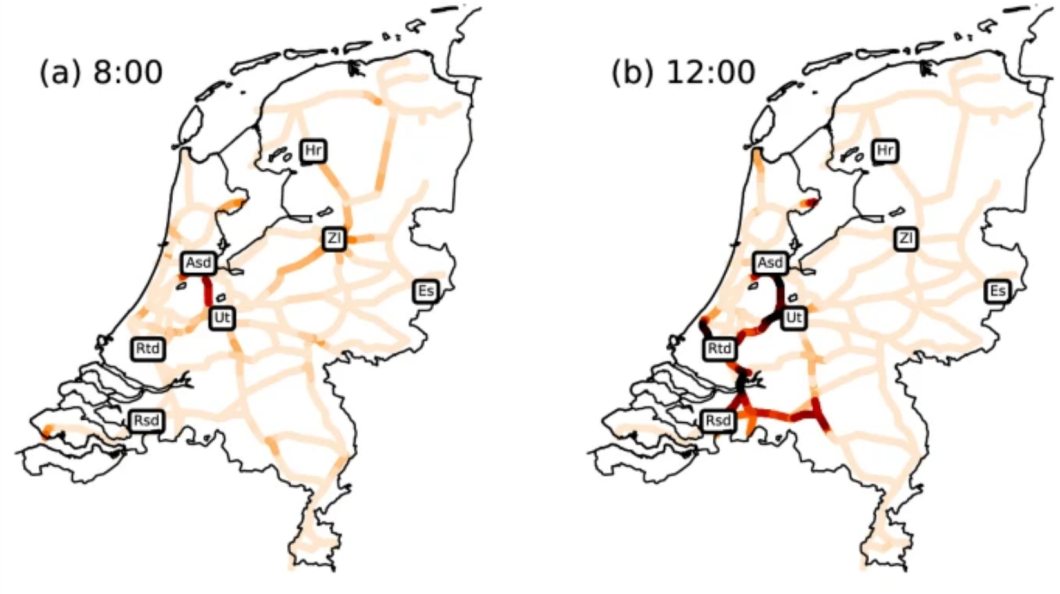

FIGURE 1.4: Propagating delay on 3 February 2012. There were many out-of-control situations due to extreme weather conditions. [Dekker et al. 2021]

Train dispatchers have knowledge about their own region and mainly see the results of delay propagation within their own area. They have less knowledge about how delays will propagate towards their region. The consequence of this limited area is that they do not have a global picture of the disturbances. It may occur that the solution for their own area will reduce the amount of disturbances, but that this solution does not reduce the impact on other regions [Dekker et al. 2021].

The train schedule of the Netherlands is dense, and delays propagate fast. When something unexpected happens, the train dispatcher has to work under a high time pressure to reduce the impact of the occurrence. There are different options for handling a delayed train. Currently, the traffic controllers make these decisions to minimize the disruption based on expert knowledge and rule based procedures (TAD). A decision support system could help the train dispatcher with these decisions. The goal of this thesis is to

find causal relationships between interacting trains; this will in particular enable us to estimate train delays.

1.4 Context of project

This project is a component of a larger project of ProRail and University of Utrecht. The goal of this thesis is the exploration of causal analysis applied to delay data, which could be a precursor to a decision support system that minimizes the disruption when a train gets delayed. If a delay for a train arises, it not only affects that train, but a chain reaction of other delays propagate to the non-delayed trains. The main goal of this thesis is to learn a Structural Causal Model (SCM) that models the system for a certain day, using ProRail data. The current implementation of the delay prediction system of ProRail does not include railway interaction of trains in their model. This research focuses on delays caused by train interaction on the railway, which is called a train-train delay. Delays due to shortage of train drivers or train conductors are therefore not included in our model, but treated as noise in our model.

There is a big difference between delay prediction of a passenger train and a freight train. Passenger trains have their predetermined recurring schedule and drive almost every day at the same time slots. Freight trains don't drive according to a recurring schedule, but they drive demand driven [ProLeren n.d.(b)]. This means that the data for freight trains does not have the same repetition every day or week, and it is difficult to learn from inconsistent data. This research will therefore only consist of predicting the delays for passenger trains. The freight trains that interact with passenger trains are treated as noise in our model. The percentage of freight trains on the railway varies across different areas in the Netherlands, but in our datasets, this ranged from 18.72% to 27.9%.

1.5 Research question(s)

The objective of this thesis is to build a causal model of interacting trains, which can enable us to improve the prediction of delay propagation. This leads to the following research question:

How to derive a causal graph from train network data in the presence of background knowledge and learn the assignment functions?

This research consists of two parts, the structure discovery and the equation discovery part. The structure discovery part consists of a hybrid method that combines background knowledge and a causal discovery method. The equations relating the variables will be learned by implementing multiple Neural Networks.
The main research question will be answered by addressing the following subquestions. At every subquestion, action points are described to answer this question.

1. *What does the train network data look like?*
    1.1. Look into the possibilities of the available data to find the important columns that can be included for this research.
    1.2. Determine the noise distribution of the delay variable by consulting an expert at ProRail, plotting the distributions, and if necessary, approximate the distribution.
2. *Which methods can be used to create a SCM graph for the train network data?*
    2.1. Perform a literature study to identify the current methods to create an SCM graph from data.
    2.2. Determine which expert knowledge can be used that benefits the causal discovery accuracy and computation speed.
    2.3. Implement and test the found solutions and, if necessary, improve the causal discovery phase
3. *How can the assignment functions be learned using Neural Networks?*
    3.1. Perform a literature study on the current methods to create a Neural Network to predict delays.
    3.2. Determine which kind of input variables are important, by performing a literature study and testing different configurations by trial and error

3.3. Determine which kind of loss function is suitable, by determining the distribution of the data and finding or calculating a corresponding loss function.

3.4. Determine which kind of activation functions are suitable, by performing a literature study about existing activation functions and testing by trial and error.

4. *How can the created SCM be evaluated?*

4.1. Perform a literature study about suitable evaluating methods

4.2. Determine how accurate the SCM is by implementing the chosen method(s)

Chapter 2

Train related background information

In this section, we will discuss the most important terms in the railway world that apply to this research.

Train series and train number
The train series denotes a fixed train route that is present in the Netherlands and has fixed stations where it stops (e.g., 7400 denotes the route from Uitgeest to Rhenen, via Utrecht, in both directions). It is comparable to the definition of a bus line. A route will be driven multiple times a day, so per train, there is a unique train number assigned to identify a specific train. The train numbers are based on the train series. Train numbers of for example train series 7400 are ranging from 7401 to 7499, such that the train series is still derivable. The even numbers denote the rides from one direction and the odd numbers denote the rides from the other direction.

Activities of trains
Within the data, the activity of each train is captured at certain points on the track. There are 6 types of activities, K_V, K_A, A, V, D and R. K_V and K_A stand for Kort Vertrek (*Short Departure*) and Korte Aankomst (*Short Arrival*). The train is then planned to stay here for maximal one minute. A and V stand for Aankomst en Vertrek (*Arrival and Departure*): these activities take longer than one minute. D stands for Doorkomst (*Passage*), which means that the train does not stop at that point. R stands for Rangeren (*To shunt*), which means that the train is placed to another part of the track to construct a new train or to remove (a part of) the train [ProRail n.d.(b)].

Train type

There are different types of trains, such as IC, SPR, ICE, THA, HSN and LM. One distinction can be made between the intercity (IC) and the sprinter (SPR). The intercity only stops at larger stations, while the sprinter also stops at intermediate stations. The international trains InterCity Express (ICE) travels to Germany and the Thalys (THA) travels to France. The HSN is the Dutch in-country train on the high speed railway. LM stands for leeg materiaal (English: Empty equipment), and are the trains used for shunting.

Dienstregelpunt (drp)/ timetabling point

A dienstregelpunt is a measurement location on the railway. The train schedule is created by determining all planned times for which the train has to arrive, departure or pass though at the dienstregelpunten. A station is also a type of dienstregelpunt, which may contain two activities: arriving and departing. In this research, we further refer to the timetabling point as the abbreviation *drp*.

Non-conflicting route

The train dispatcher can only enable a route for a train when, among other things, there is a non-conflicting route. A non-conflicting route means that there is no train is on the desired section or planned to be on that section. In Figure 2.1 an example of a part of the Operator's Control Unit of the train dispatcher is shown. Three trains are shown, denoted with a number, and located on the yellow line. The yellow line indicates that the train is at this section. The routes that are enabled, are denoted as a green line, and an arrow on that line denotes the direction of the train [ProRail n.d.(b)]. If then a new train has to enter this section in the image, a fourth route has to be enabled. This route cannot cross or contain a part of the green lines, since this route is then conflicting.

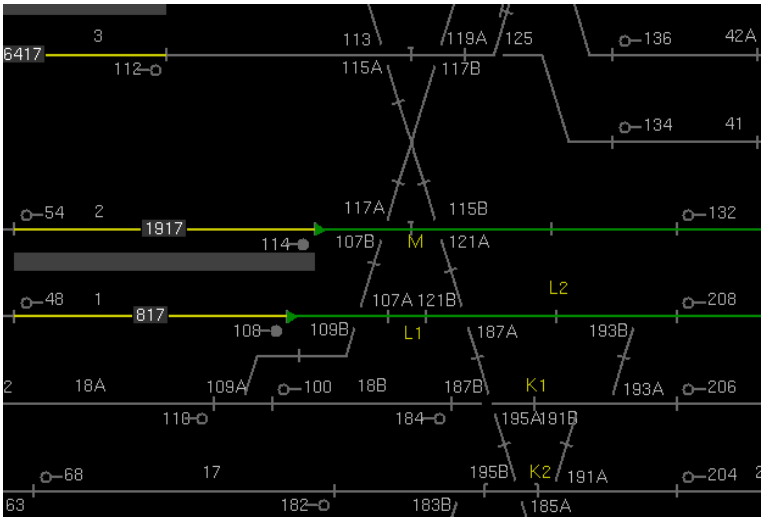

FIGURE 2.1: Example of one of the Operator's Control Units ProRail (n.d.(b))

The open track and the yard

On the open track, there are no controllable switches or signs. At the yard, there are controllable switches and signs.

Dwell time, Buffers

Dwell time is the time difference between arrival and departure of a train at a station. Within this time, at least five processes are performed, namely door unlocking, door opening, boarding, door closing and train dispatching [Li, Daamen, et al. 2016]. This is the minimal time that a train has to wait at the station. At some stations, the train waits longer than only that minimal necessary time. The remaining time functions as a buffer: small delays can be resolved and do not propagate during the rest of the shift.

Train-train delay

A train delay that has an external cause, such as the weather or engine problems, is called a primary delay. The delays that arise from this primary delay are called the secondary delays or knock on delays. An example of a knock on delay is a delay caused by railway interaction: a delayed train occupies the railway at a different time than planned, requiring another train to wait. Another example of a knock on delay is that the train driver or conductor has to transfer to another train, but are delayed. This results that the train he/she transfers to will also be delayed. In this research, we only focus on delays that occur due to railway interaction. This means that the delay of a train is caused by a previous delayed train. We denote this as a train-train delay.

Causes of train-train delays

This research only focuses on the train-train delays. We can distinguish between different train-train delays.

1. **Same train** A train that is delayed, will probably also be delayed at its next stops. The delay, therefore, propagates to the same train on different stations. We denote this as train_x at station_i influences train_x at station_{i+1}. [Corman et al. 2018].

2. **Non-conflicting route** A train can only drive if there is a non-conflicting route enabled. If a train has a large delay, some parts of the route are not enabled yet, since the train has not arrived yet. When the delayed train has arrived, the non-conflicting route would be enabled at another time than planned. It can occur that the non-conflicting route would then be used for another train. The train dispatcher can decide two things:

   1. The other train uses the non-conflicting route first, so the delayed train gets more delay.

   2. The delayed train uses the non-conflicting route first, but the other train now gets a delay as well.

3. **Transfer** At some stations, the planning is made such that there are transfers possible between different trains. If a train X is delayed and the transfer is not possible anymore, some trains Y may wait to still enable this transfer. Then train X causes a delay on Y. However, if the delay of train X is too large, train Y will depart without delay because else train Y will have a delay that is too large [ProLeren n.d.(c)].

4. **Change of train number** When a train has fulfilled its route, it can be deployed as another train to drive another route. If a train has a delay when finishing its route and is deployed to drive the next route, this route could have a delay at the start. This is why at the endpoints of routes there is a large buffer calculated, to prevent such situations.

Chapter 3

Literature review

This section provides an overview of the current train delay prediction methods, focusing on the Linear Regression method, Bayesian Networks and Neural Networks. It further describes which features were included in prediction models.

3.1 State of the art of train delay prediction methods

Spanninger et al. (2022) provides a review of the different methods to predict train delays. They distinguish between event driven and data driven methods. Event driven methods take dependencies of train events into account when modeling, which results into multistep predictions. If we want to know what happens for train T at station $i+3$, the model first calculates the intermediate steps, so $x_i \rightarrow x_{i+1} \rightarrow x_{i+2} \rightarrow x_{i+3}$. The data driven method calculates the delay directly, so in this case $x_i \rightarrow x_{i+3}$. Figure 3.1 provides an overview of the existing models for delay prediction, and the models are classified between event driven or data driven approaches and between stochastic models and deterministic models [Spanninger et al. 2022]. This section further elaborates on existing methods to predict train delays using Linear Regression, Bayesian Networks and Neural Networks.

3.1.1 Delay prediction using Linear Regression

A Linear regression model can be used to predict delays. Explanatory variables such as train number of crossings, speed limit and boarding time are given as input, and a delay as output variable is returned. Murali et al. (2010) introduce a simulation-based method to estimate train delays using a linear regression model. The purpose for this research is to schedule freight trains over large networks and estimating the delay.

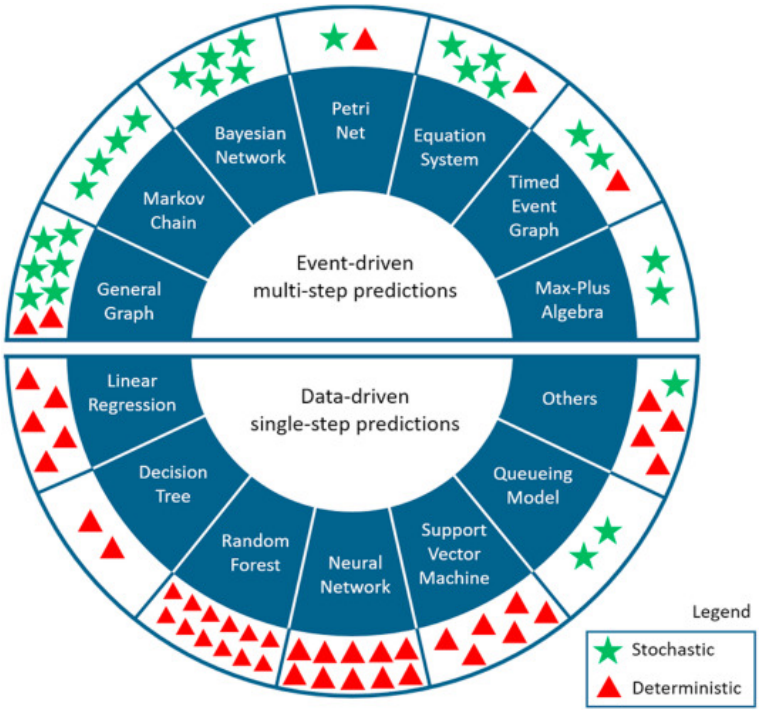

FIGURE 3.1: Overview of existing methods for delay prediction [Spanninger et al. 2022]

Wen, Li, et al. (2017) used a linear regression model to do analysis on the primary delay of a HSR line in China. They investigated among other things the temporal distribution of primary delays and concluded that during peak time more primary delays occur. In Figure 3.2 the duration of the primary delay is plotted against the frequency, for which they conclude that for this distribution the log-normal distribution fitted the data best. [Wen, Li, et al. 2017]

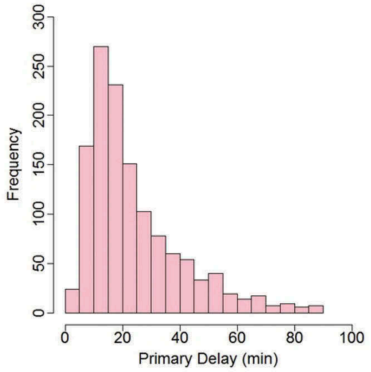

FIGURE 3.2: Primary delay vs Frequency [Wen, Li, et al. 2017]

Li, Daamen, et al. (2016) have created two different regression models to estimate the train dwell times at the short stops between Utrecht and Eindhoven. Dwell times mainly depend on the number of passengers that enter and leave the train, but these are not available real time. They therefore estimate the dwell time using a statistical analysis

with possible predictor variables. The dwell time for peak and off-peak hours are estimated by different methods. In Li, Goverde, et al. (2014) they concluded that the dwell times fit a log-normal distribution, which they have included in one of their models [Li, Daamen, et al. 2016].

3.1.2 Delay prediction using Bayesian networks

Bayesian networks are used by Zilko et al. (2016) to predict the disruption length. Lessan et al. (2019) introduce Bayesian Networks to formulate delay dependencies for the High Speed Rail (HSR) between the Chinese stations Wuhan and Guangzhou. They create three different Bayesian Network graphs and estimated the parameters using Maximum Likelihood Estimation (MLE). They created a hill climbing method and a primitive linear method. The third structure is created using a hybrid method, where they first used the hill climbing and the primitive linear method, and improved the network using domain knowledge. This model can predict with 80% accuracy the delays within a 60-minute time horizon.
Corman et al. (2018) created a hybrid method that focuses on how the accuracy of the prediction of an event changes when the time interval towards the event decreases and more information becomes available. They use domain knowledge to create their Bayesian Network structure and have included dependencies between trains that share infrastructure or have scheduled passenger transfers in their model. They limited their prediction horizon to 60 minutes, since predictions for distant events will become less accurate. The parameters of the Bayesian Network have linear coefficients, since a local distribution can be represented as a linear model according to Koller et al. (2009).

Huang, Lessan, et al. (2020) introduce a hybrid model that predicts three different consequences of a disturbance for two HSR lines in China. The primary delay L per station (which is the first delay that occurs at that station), the number of delayed trains per station N and the total delay time per station T. The authors look into the temporal and spatial propagations of delays of the train activities. The structure of the model describes the relationship between L, N and T at each station and towards the next station. They propose 4 different structures and choose the best one using comprehensive experiments. Parameter learning is done by MLE.

Another hybrid model, the Context-Driven Bayesian Network (CDBN), is proposed, which combines a k-means clustering algorithm with a Bayesian Network [Huang, Spanninger, et al. 2022]. K-means is used to find delay evolution patterns, which is the classification whether the delay of a specific train increasing, staying the same or decreasing. Per cluster, a Bayesian Network is constructed with the clustered data. Ulak

et al. (2020) also uses Bayesian Network learning to calculate the impacts of the delays at certain stops at Long Island Rail Road. They used MLE to find the parameters and max-min hill–climbing (mmhc) to learn the graph. Mmhc combines a constrained based algorithm to find the skeleton of the Bayesian Network and a greedy hill-climbing search to orient the edges [Tsamardinos et al. 2006].

3.1.3 Delay prediction using Neural Networks

The Long Short-Term Memory (LSTM) model is a popular method for delay prediction. The LSTM is an improvement of the Recurrent Neural Network, which is a Neural Network consisting of one or more cycles, such that a node 'remembers' the previous state. The LSTM contains a special memory gate such that it 'remembers' long term dependencies better [Mou et al. 2019]. Mou et al. (2019) used a LSTM model for delay prediction for a section of the Dutch railways, Rotterdam Centraal to Dordrecht. This delay prediction consists of a prediction horizon of one day and outputs the arrival delay time of the train at the next station. They state that knowledge about the schedule is not important for the neural network learning process, since current methods can learn the important features from the large amount of data. Instead, they focus on the selection of characteristic input variables. Wen, Mou, et al. (2020) continued with the previously mentioned paper and tested the LSTM prediction model further for the sections Rotterdam Centraal to Dordrecht and Rilland Bath to Vlissingen. The LSTM of Mou et al. (2019) and Wen, Mou, et al. (2020) both outperform the Artificial Neural Network and the Random Forest model.

An LSTM is also used within hybrid algorithms. Huang, Wen, Fu, Lessan, et al. (2020) created an algorithm that combines a fully-connected neural network (FCNN) and LSTM method. Not only they focus on the infrastructure-related and operational-related variables, they also included weather-related variables, such as temperature and wind speed, as input for the neural network. This additional information contributes significantly to the train prediction delays. Wu et al. (2021) creates a hybrid method that combines LSTM with Critical Point Search (CPS) for long term delay prediction. CPS is a rule-based classification method, which is used in this paper to predict the primary and secondary delays. Luo et al. (2022) proposes a Bayesian optimization-based multi-output deep learning model. This is a hybrid method that consists of the FCNN and LSTMs and outputs the delays of n subsequent trains at a specific station. The model is tested with the operational data from the Wuhan–Guangzhou HSR line. The estimation error increases with the time interval and with which sequential train (first, second etc.) it predicts. Huang, Wen, Fu, Peng, et al. (2020) proposes the CLF-Net, which consists

of a 3D Convolutional Neural Network (CNN), a LSTM and a FCNN. The CLF-Net is trained with spatio-temporal data, time series data and none-time-series data. This is trained and tested against different models, for which the CLF-Net was the best scoring model. Li, Huang, et al. (2022) proposes a similar hybrid method to predict the delays at multiline stations.

Oneto et al. (2018) used Shallow Extreme Learning Machines (SELM) and Deep Extreme Learning Machines (DELM) to create a dynamic delay prediction method. SELM is a single-hidden-layer feed forward neural network, and DELM is a feed forward neural network that consists of multiple hidden layers. The method predicts the train delay at the next checkpoint.

Feature space

Determining the feature space is an important step in delay prediction [Mou et al. 2019]. These features can include external variables: **calendar features** such as day of the week, holiday or working day [Oneto et al. 2018], **weather related features** such as wind speed or rainfall [Huang, Wen, Fu, Lessan, et al. 2020] or **infrastructure related features** such as the length of sections [Huang, Wen, Fu, Lessan, et al. 2020], speed limit number or crossings [Murali et al. 2010].

Most studies included train operation variables of the predicted train such as the train type (IC, SPR, LM) [Mou et al. 2019] [Wu et al. 2021], dwell times and running times [Oneto et al. 2018] [Wu et al. 2021] [Huang, Wen, Fu, Lessan, et al. 2020] or arrival and departure delay at the current or previous station [Mou et al. 2019] [Huang, Wen, Fu, Lessan, et al. 2020].

Some studies also included features of other interacting trains within a time interval, such as running times or dwell times [Oneto et al. 2018] or delays [Huang, Wen, Fu, Lessan, et al. 2020]. Some models use features of m trains previous and subsequent of a delayed train at a certain station [Wu et al. 2021]. Huang, Wen, Fu, Lessan, et al. (2020) only look at the previous planned train on a section.

Chapter 4

Introduction to SCMs

A Structural Causal Model is a model that consist of variables and assignments formulas which represents the causal relationships between variables in a system. An SCM graph represents this visually: a directed edge between two variables $X \rightarrow Y$ indicates that X is a cause of Y. It is difficult to model an interaction of the world with an exact outcome, since there are external influences. To denote external influences, *exogenous* variables are introduced, which represents the influences outside the model. The variables that are modelled, are called *endogenous* variables [Pearl et al. 2016].

Conditional independence is defined as follows. If we have variables A B and C with the probabilities $P(C|A,B) = P(C|B)$, we can state that C is conditionally independent of A given B. Implicating, A does not provide any new information for C when B is known. This independence can be written symbolically as $(C \perp\!\!\!\perp A \mid B)$ [Pearl et al. 2016]. An independence test can be performed to determine if there is a correlation between two variables. This is a statistical testing method to assess whether two variables are independent of each other; if this is not the case, the two variables are correlated.

Paths are present within a graph, which can be blocked or unblocked. If we have a chain $A \rightarrow B \rightarrow C$ or a fork $A \leftarrow B \rightarrow C$, we have an unblocked path. When conditioned on B, this path is blocked. For a collider $A \rightarrow B \leftarrow C$, this path is automatically blocked, unless we condition on B. If a path between X and Y in a graph is blocked by a set Z, we state that X and Y are **d-separated** conditional on Z. [Pearl et al. 2016]

Within the graphical models, the Markov properties are an important aspect. It consists of different properties, namely the Global Markov property, the Local Markov property and the Markov factorization property. The **Global Markov Property** states that if

there is a d-separation in graph G: $X \perp\!\!\!\perp_G Y \mid Z$, there is a conditional independence in probability model P: $X \perp\!\!\!\perp_P Y \mid Z$. The **Local Markov property** states that each variable is independent of its non-descendants when conditioning on its parents. The **Markov factorization property** states $P(x_1, .., x_k) = \prod_{i=1}^{k} P(x_i | pa_i^G)$ [Peters et al. 2017].

Another definition is the **Markov equivalence**. Two graphs are Markov equivalent if they have the same skeleton *and* they have the same set of v-structures. A v-structure is a collider for which its parents are not adjacent [Pearl et al. 2016]. The inverse of the Global Markov Property is **Faithfulness**. This states that a conditional independence in model P implies a d-separation in graph G: $X \perp\!\!\!\perp_P Y \mid Z \Rightarrow X \perp\!\!\!\perp_G Y \mid Z$. [Peters et al. 2017].

4.1 Causal discovery methods

To derive a SCM graph from data, many algorithms have been created. This section describes different types of causal discovery algorithms, which assumptions they have, and what their limits are. There are two main algorithms, the constraint-based algorithms and the score-based algorithms.

Constraint based algorithms perform statistical tests, such as a conditional independence test, on a dataset and remove the edges between nodes that are proven to be independent. It returns a Partial Ancestral Graph (PAG), the Markov Equivalent class of the graphs, that satisfies these tests [Triantafillou et al. 2016] [Ogarrio et al. 2016]. Score-based methods use a scoring function that determines how well the graph reflects the data to find the best graph that maximizes the score [Triantafillou et al. 2016].

4.1.1 Peter-Clark (PC) algorithm

The PC-algorithm is a constraint based algorithm and has four steps [Spirtes et al. 2000]:

1. *Start with the complete undirected graph.*
2. *Skeleton search by performing independence tests on the data.* When an independency between two nodes is found, the separating set of these nodes is updated with the

set that is conditioned on. The edge between the two nodes is removed, and the graph remains undirected.

3. *Identify the v-structures and orient those edges.* The v-structures can be identified by looking at the separating set. If there is a v-structure $A \rightarrow B \leftarrow C$ in the true graph, we must find in the data that A en C are dependent given B. Then B is not in Sepset(A,C) and we can conclude that the structure of A,B and C is $A \rightarrow B \leftarrow C$.

4. *Orient other edges.* When knowing some directed edges, other undirected edges may be oriented as well. If we know for example $A \rightarrow B - C$, we should have found $B \leftarrow C$ if there was a v-structure present. Since we did not find this structure in step 2, we must conclude that the correct graph is $A \rightarrow B \rightarrow C$.

The complexity of the PC-algorithm is bounded by the maximal degree k (maximal neighbors of a node) of the true-graph and the number of nodes n. The amount of independence test needed is then in the order of $2\binom{n}{2}\sum_{i=0}^{k}\binom{n-1}{i} = \frac{n^2(n-1)^{k-1}}{(k-1)!}$ [Spirtes et al. 2000].

The PC- algorithm has the following assumptions:
1. Sufficiency assumption, there are no unknown confounders [Glymour et al. 2019]
2. The Local Markov property holds [Glymour et al. 2019]
3. Faithfulness assumption [Glymour et al. 2019]
4. The causal graph is a directed acyclic graph (DAG)

4.1.2 Fast Causal Inference (FCI)

The FCI algorithm is a variation of the PC-algorithm. An improvement of this algorithm regarding the PC-algorithm is that it does not assume the absence of unobserved confounders in the data, and can even detect them. These are marked in the graph with a bidirectional edge between two measured variables, meaning that there is an unmeasured confounder present [Glymour et al. 2019]. To relax the sufficiency assumption, the algorithm needs to test for independencies between non-adjacent variables, which results into the possible d-separating sets [Raghu et al. 2018]. The FCI algorithm works partially the same as the PC algorithm, but with some extra steps [Spirtes et al. 2000]:

1. *Start with the complete undirected graph.*

2. *Skeleton search.* Do skeleton search by performing independence tests and keeping track of the SepSet for each edge. Edges between nodes that are proven to be independent are removed.

3. *Orient edges based on v-structure.* Orient the remaining edges of the graph as undirected in both directions (denoted as o–o) and identify the v-structures and orient those edges. This results into graph F.

4. *Extra separation step.* The remaining edges are pruned further by looking if a d-separation between two variables is possible, given a subset of the graph. The FCI algorithm keeps track of the Possible-D-SEP(X,Y) S of graph F, by including only the variables that *may* separate X and Y. If X and Y can be separated by any subset of S in graph F, the edge between X and Y is removed and the separating sets of X and Y are set to S.

5. *Orient edges based on v-structure.* Since there now is a different graph created in step 4, all edges are unoriented again and the v-structures are identified.

6. *Orient other edges.* The remaining edges can be oriented with the same approach as described in step 4 of the PC-algorithm.

4.1.3 Greedy Equivalence Search (GES)

This algorithm is a score based algorithm and searches greedily for a better scoring graph. It starts with the forward stage: the empty graph is constructed and edges are added to the graph as long as the score function improves. In the backward stage, edges are removed as long as the score function improves. There are different scoring functions, such as the Bayesian Information Criterion (BIC) score, which combines a likelihood score with a penalty for the complexity [Chickering 2002]. Each score has its own assumptions: the BIC and the BGe score assume the noise follows a normal distribution. The BDeu/BDe scores are meant for discrete data [Huang, Zhang, et al. 2018]. The GES algorithm assumes that there are no unmeasured confounders, the Local Markov property holds, the faithfulness assumption holds, and the causal graph is a DAG [Ogarrio et al. 2016].

Comparing GES and the PC-algorithm, the quality of the estimations of GES is better, but has a slower performance than PC [Nandy et al. 2018]. The Fast Greedy Search (FGS) algorithm is a modification of GES, which works the same as GES, but uses a different data structure, which results into a faster algorithm [Ogarrio et al. 2016]. Just as with GES, the FSG algorithm may include too many edges, since it has the assumption that there are no unmeasured confounders present.

4.1.4 GFCI

GFCI is a hybrid algorithm, since it combines the constraint based FCI algorithm and the score-based FGS algorithm. GFCI first applies the FGS algorithm and uses the outcome as the skeleton for the FCI algorithm. Comparing GFCI and FCI, GFCI has a lower average estimation error and a better recall and precision score for finding adjacencies and identifying the endpoints of the edges than FCI. GFCI does have a longer computation time than FCI [Ogarrio et al. 2016]. Since the FCI algorithm is applied, this algorithm does not assume the absence of latent confounders.

Chapter 5

Overview of the steps of the complete method

In this section, a step-by-step roadmap is presented for the creation of our prediction model. The code can be found at https://github.com/vera98x/Causal-discovery. Our goals are: finding the causal relations between trains, and training the Neural Networks using the causal relations and the external variables to estimate the delays. The roadmap of our method is illustrated in Figure 5.1.

The first step is to clean the initial dataset, such that each day consists of the same events. This is described in Section 6.2. Then, two TrainRideObjects (TRO) matrices are built: the TRO schedule and the TRO matrix. A TRO stores all the relevant information of a specific event, such as train series, train number, drp, platform, delay, planned time, etc. The TRO schedule only contains the schedule, which is a list of all TROs that occur in one day. The other is the TRO matrix that denotes all the events that occurred in our dataset. The rows denote the different days and the columns denote the different events. Section 6.3 elaborates on this further.

The TRO schedule is used as input to create the background knowledge, as described in Section 7.2.1. To find the causal graph, the background knowledge and the TRO matrix are provided. The TRO matrix is beforehand converted to a matrix of integers that denote the delay per event. Section 7.2.2 elaborates on how the causal graph is retrieved.

The input for the Neural Networks consist of information about the causal relations between trains, provided by the causal graph, and the external variables, provided by the TRO matrix. Using this input, the pre-trained Neural Network and the multiple fine-tuned Neural Networks are created. This is explained in Section 8. The details

about how the graph and the TRO are converted to the input for the Neural Networks are explained in Appendix G.

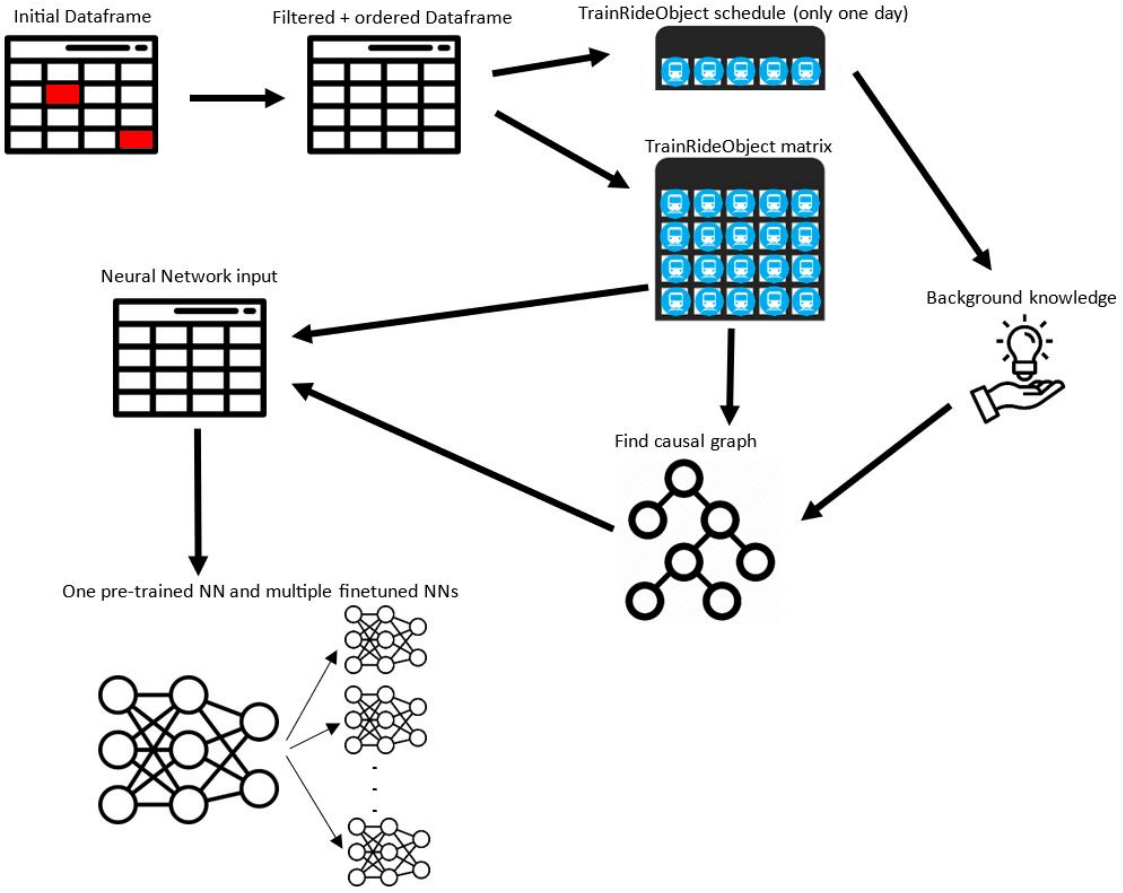

FIGURE 5.1: Roadmap of our method

Chapter 6

Data description

The data is retrieved from an internal system of ProRail, which contains historical data of the punctuality of the train rides. For this research, three areas of the Dutch railway system are used. In Figure 6.1, these areas are colored in the map. The areas are Amsterdam—Utrecht (red frame), Utrecht-Eindhoven (blue frame) and Rotterdam-Dordrecht (purple frame). All datasets include the activities of the trains that were present between September 4, 2017 to December 8, 2017 (70 working days)

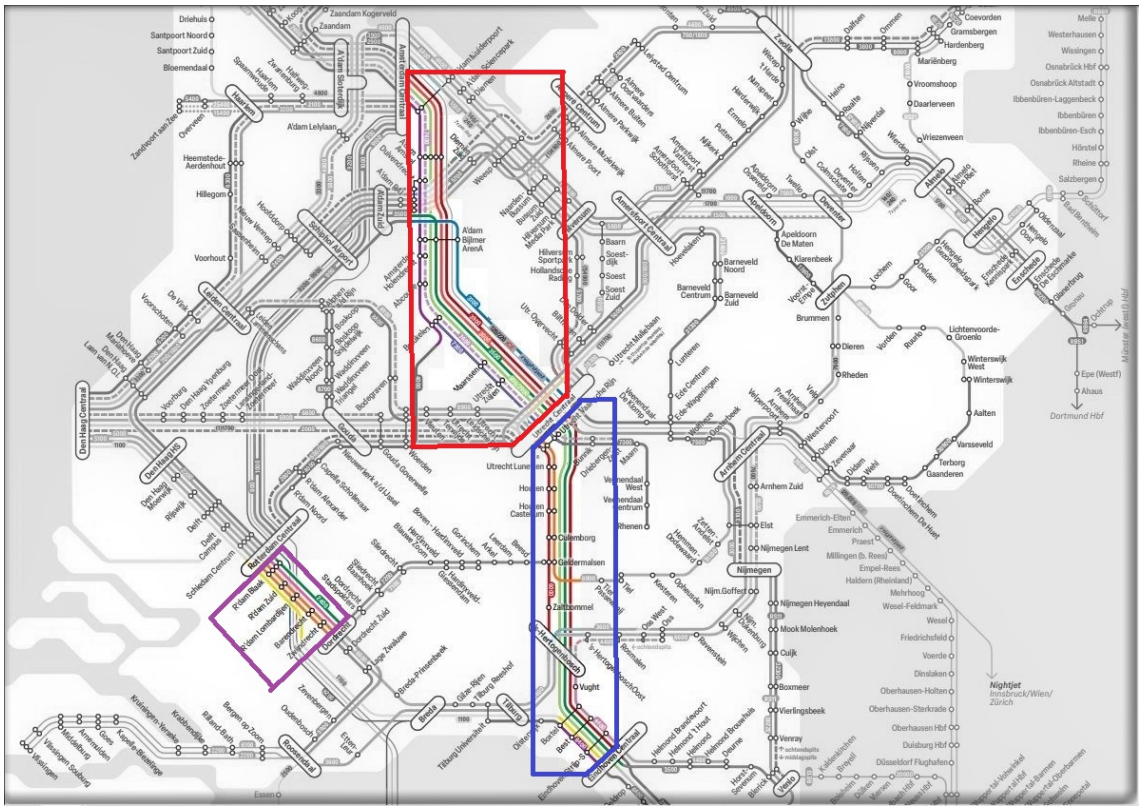

FIGURE 6.1: Railway map with the predicted sections colored

6.1 Description data sets

The railway between Rotterdam and Dordrecht consist of 12 timetabling points, including 7 stations (printed in bold), namely **Rotterdam Centraal (Rtd)**, Willemstunnel noord (Wiltn), **Rotterdam Blaak (Rtb)**, Willemstunnel zuid (Wiltz), **Rotterdam Zuid (Rtz)**, Rotterdam Stadion (Rtst), **Rotterdam Lombardijen (Rlb)**, **Barendrecht (Brd)**, Kijfhoek aansluiting Zuid (kfhaz), **Zwijndrecht (Zwd)**, Grote Brug (Grbr) and **Dordrecht (Ddr)**.

The railway between Amsterdam and Utrecht consist of 18 timetabling points, including 11 stations (printed in bold): **Amsterdam Centraal (Asd)**, Oosterdoksluis (Ods), Amsterdam Dijksgracht Westzijde (Dgrw), Amsterdam Muiderpoort Aansluiting (Asdma), **Amsterdam Muiderpoort (Asdm)**, **Amsterdam Amstel (Asa)**, **Duivendrecht (Dvd)**, Duivendrecht aansluiting Zuid (Dvaz), **Amsterdam Bijlmer (Asb)**, **Amsterdam Holendrecht (Ashd)**, **Abcoude (Ac)**, Abcoude overloopwissels (Aco), **Breukelen (Bkl)**, Breukelen aansluiting (Bkla), **Maarssen (Mas)**, **Utrecht Zuilen (Utzl)**, Utrecht Maarssen aansluiting (Utma) and **Utrecht Centraal (Ut)**.

The railway between Utrecht and Eindhoven consist of 23 timetabling points, including 14 stations (printed in bold): **Utrecht Centraal (Ut)**, **Utrecht Vaartsche Rijn (Utvr)**, Utrecht Zuid aansluiting (Utza), **Utrecht Lunetten (Utln)**, **Houten (Htn)**, **Houten Castellum (Htnc)**, Lekbrug (Lek), **Culemborg (Cl)**, **Geldermalsen (Gdm)**, Meteren Betuweroute aansluiting noord (Mbtwan), Meteren Betuweroute aansluiting zuid (Mbtwaz), **Zaltbommel (Zbm)**, Oud Zaltbommel (Ozbm), Hedel (Hdl), **'s-Hertogenbosch (Ht)**, Vught aansluiting (Vga), **Vught (Vg)**, **Boxtel (Btl)**, Liempde (Lpe), Best overloopwissels (Beto), **Best (Bet)**, Acht (At), **Eindhoven Strijp-S (Ehs)**, and **Eindhoven (Ehv)**.

An example of rows within such dataset is denoted in Table 6.1. Each row consists of one event (arrival, departure, or passage) that occurred for a specific train. In this example, 7 columns are shown, but in reality there are more columns present. The complete overview of columns is denoted in Appendix A

Date	Train series	Train number	Drp	Activity	Planned time	Realized time
09-09-2017	1100E	1122	Rtz	D	08:43:24	08:43:43
25-09-2017	5100E	5174	Zwd	V	21:15:00	21:14:50
15-11-2017	2200O	2257	Rtb	K_A	15:05:06	15:05:12
07-12-2017	5000O	5065	Ddr	A	18:04:00	18:03:35

TABLE 6.1: Example of rows in the dataset

6.2 Preprocessing steps

The algorithm that is used to find causal relationships uses a $n \times m$ matrix with n rows and m columns as input. The algorithm is described in section 7.2. Each column represents an event (e.g., train 7424 arriving at station Bkl) and each row represents a day. This means that for every day, the same events need to be included. In the dataset, this is not the case. For example, more train rides occur on one day of the week than on the other. Further, sometimes trains are randomly added to the daily schedule and sometimes a train gets cancelled. Therefore, it is important to do some preprocessing to include the same events for each day. The dataset is further preprocessed by performing the following action points in this order. After these steps, the resulting Dataframe of c columns is a $(n \cdot m) \times c$ matrix. Each row still consist of one activity. The transformation to a $n \times m$ matrix is done in another step, described in section 6.3. Every step described below is illustrated afterward.

1. **Only keep the useful columns** The dataset consists of many columns, including columns that are not directly used for the delay prediction program. However, these columns do say something about the data. For example, there is a column such that we can derive the amount of interventions that are done. The columns that are not used in the program are discarded. The kept columns are assigned a type and some columns are renamed.

2. **Remove the duplicated items** Sometimes an event is registered twice. We then only keep the first row of this event and delete the other one(s).

3. **Create new columns using the other columns** By combining other columns, new columns are created, such as delay, buffer time and travel time. This new information will be added as external information for the Neural Network. Specifically, the variable "delay" is used to learn the causal graph as well.

4. **Remove columns that do not contain a date** If the date column is empty, this row is removed. Another solution would be to look at the planned time, which also includes a date, but these are not always the same. Night trains are grouped by the previous date, so errors will be made when using this imputation method.

5. **Only keep workdays** The schedule between working days are similar to each other, but the schedule differs much on the weekend. We therefore only include working days in our research.

6. **Remove cancelled trains** Before finding the schedule, remove the cancelled train. This prevents that trains that are always cancelled are not included in the schedule.

7. **Find the schedule** The data is grouped by train number, drp, activity and the planned time. The grouped events that do not occur above a certain threshold, are filtered out of the dataset. Using the remaining events, the schedule is created. Lastly, the trains are removed from the schedule that never have a basic_uitvoer.

8. **Modify data according to the schedule** Using this created schedule, the data set is modified such that every day consist of the same events. That means: removing every event that is not in the schedule, and adding events that are in the schedule and mark them as a cancelled train for that day.

These steps are further illustrated through an example. An initial (fake) dataset is presented, and at each step, the data set is modified accordingly. In row 10, the planned time is marked red, indicating that this train was cancelled. The column "sections driven" is not used in the model, thus considered an unnecessary column. The initial table is denoted in 6.2

Index	**Date**	**Train number**	**Drp**	**Activity**	**Planned time**	**Realized time**	sections driven
1	24-09-2017	7424	Blk	K_A	08:44:24	08:44:50	A to B
2	24-09-2017	7424	Blk	K_V	08:43:24	08:43:30	C to D
3		7424	Blk	K_A	08:43:24	08:43:43	A to B
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50	C to D
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30	A to B
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50	C to D
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20	X to V
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:20	A to B
9	27-09-2017	7424	Blk	K_A	08:43:24	08:43:20	A to B
10	27-09-2017	7424	Blk	K_V	08:44:24		C to D

TABLE 6.2: Example of rows in the dataset, initial table

In step 1, the unnecessary columns are removed and the column 'sections driven' is removed, since this column will not be used. This results in Table 6.3.

Index	**Date**	**Train number**	**Drp**	**Activity**	**Planned time**	**Realized time**
1	24-09-2017	7424	Blk	K_A	08:44:24	08:44:50
2	24-09-2017	7424	Blk	K_V	08:43:24	08:43:30
3		7424	Blk	K_A	08:43:24	08:43:43
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:20
9	27-09-2017	7424	Blk	K_A	08:43:24	08:43:20
10	27-09-2017	7424	Blk	K_V	08:44:24	

TABLE 6.3: Example of rows in the dataset, step 1

In step 2, the duplicated items are removed. The row with index 9 is deleted and this results in Table 6.4

Index	Date	Train number	Drp	Activity	Planned time	Realized time
1	24-09-2017	7424	Blk	K_A	08:44:24	08:44:50
2	24-09-2017	7424	Blk	K_V	08:43:24	08:43:30
3		7424	Blk	K_A	08:43:24	08:43:43
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:20
10	27-09-2017	7424	Blk	K_V	08:44:24	

TABLE 6.4: Example of rows in the dataset, step 2

In step 3, new columns are created. In this example, only the delay column denoted in seconds is added, resulting in Table 6.5.

Index	Date	Train number	Drp	Activity	Planned time	Realized time	Delay
1	24-09-2017	7424	Blk	K_A	08:44:24	08:44:50	26
2	24-09-2017	7424	Blk	K_V	08:43:24	08:43:30	6
3		7424	Blk	K_A	08:43:24	08:43:43	19
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50	26
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30	6
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50	146
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20	-4
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:00	-24
10	27-09-2017	7424	Blk	K_V	08:44:24		

TABLE 6.5: Example of rows in the dataset, step 3

In step 4, the columns that do not contain a date are removed, so the row with index 3 is removed. This results in Table 6.6

Index	Date	Train number	Drp	Activity	Planned time	Realized time	Delay
1	24-09-2017	7424	Blk	K_A	08:44:24	08:44:50	26
2	24-09-2017	7424	Blk	K_V	08:43:24	08:43:30	6
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50	26
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30	6
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50	146
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20	-4
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:00	-24
10	27-09-2017	7424	Blk	K_V	08:44:24		

TABLE 6.6: Example of rows in the dataset, step 4

In step 5, only the columns that consist of a workday are kept. 24-09-2017 is a Sunday, so rows at index 1 and 2 are removed (Table 6.7)

Index	Date	Train number	Drp	Activity	Planned time	Realized time	Delay
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50	26
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30	6
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50	146
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20	-4
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:00	-24
10	27-09-2017	7424	Blk	K_V	08:44:24		

TABLE 6.7: Example of rows in the dataset, step 5

In step 6, the cancelled trains are removed. The row on index 10 is removed, resulting in Table 6.8

Index	Date	Train number	Drp	Activity	Planned time	Realized time	Delay
4	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50	26
5	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30	6
6	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50	146
7	27-09-2017	7424	Aco	D	08:35:24	08:35:20	-4
8	27-09-2017	7424	Blk	K_A	08:43:24	08:43:00	-24

TABLE 6.8: Example of rows in the dataset, step 6

Based on the resulting table, the schedule is created in step 7, as depicted in Table 6.9. Only events that occur frequently in the data set (more than 35% of all days) are kept. The date is reset to "01-01-2000". The purpose of using the date "01-01-2000" is that it serves as a random day that is not included in the dataset and can be easily distinguished. The column "realized time" is still present, but will not be used further. Lastly, the "delay" column is empty, since these are not relevant for a schedule. Furthermore, when imputing values from the schedule in the dataset, the delay for that particular sample will automatically be left empty, as intended.

Date	Train number	Drp	Activity	Planned time	Realized time	Delay
01-01-2000	7424	Blk	K_A	08:43:24	08:43:30	
01-01-2000	7424	Blk	K_V	08:44:24	08:44:50	

TABLE 6.9: Schedule created based on the table of step 6

In the last step, the created schedule is used to modify the table from step 6, resulting in Table 6.10. The date of the added rows are marked bold and row 7 is removed, since that event is not present in the schedule. The indexes are regenerated and range from 1 to 6.

Index	Date	Train number	Drp	Activity	Planned time	Realized time	Delay
1	**25-09-2017**	7424	Blk	K_A	08:43:24	08:43:30	
2	25-09-2017	7424	Blk	K_V	08:44:24	08:44:50	26
3	26-09-2017	7424	Blk	K_A	08:43:24	08:43:30	6
4	26-09-2017	7424	Blk	K_V	08:44:24	08:46:50	146
5	27-09-2017	7424	Blk	K_A	08:43:24	08:43:00	-24
6	**27-09-2017**	7424	Blk	K_V	08:44:24	08:44:50	

TABLE 6.10: Example of rows in the dataset, step 8

6.3 Dataframe to TrainRideObject matrix

As stated in Section 6.2, it is important for the causal algorithm to have a $n \times m$ matrix as input, with n rows denoting the dates and m columns denoting the daily events. The resulting Dataframe from Section 6.2 consist of a $(n \cdot m) \times c$ matrix, where the rows denote one event and the columns denote the details of that event (e.g., the planned time or train number). The goal of the next step is to change the current matrix to a $n \times m$ matrix, where the current c columns are stored in a TrainRideObject (TRO). The TRO stores all the relevant information of a specific event, such as train series, train number, drp, platform, delay, planned time, etc.

An example of such $n \times m$ matrix can be given by proceeding with Table 6.10 from the previous section. After applying this to TrainRideObjects, the result would be as illustrated in Table 6.11. A TrainRideObject is denoted as *"TRO_..."*.

Date	7424_Bkl_K_A	7424_Bkl_K_V
25-09-2017	TRO_7424_Bkl_K_A	TRO_7424_Bkl_K_V
26-09-2017	TRO_7424_Bkl_K_A	TRO_7424_Bkl_K_V
27-09-2017	TRO_7424_Bkl_K_A	TRO_7424_Bkl_K_V

TABLE 6.11: Example of the TrainRideObject matrix

To achieve this result, the Dataframe is sorted by date, basic_treinnr_treinserie, basic_treinnr and basic_plan, and then grouped by date. Beforehand, an empty matrix of m columns and n rows is created. Per date, the rows are filled. Each date contains the same amount of activities, due to the preprocessing in Section 6.2, so each event is located at the same index per date. Per date i and event j, the element$[i,j]$ of the empty matrix is filled with a TrainRideObject. After this step, the $n \times m$ matrix is created.

Chapter 7

Finding causal relations

In this section, different types of background knowledge are proposed to incorporate into the causal discovery method. The first section (7.1) describes the domain knowledge, which is based on several assumptions. The second section (7.2) describes a hybrid method, which uses the domain knowledge in combination with a causal learning algorithm.

7.1 Domain knowledge

In this section, we explore how the causal graph can be estimated by only using the domain knowledge. A benefit of estimating the causal graph using domain knowledge instead of a causal discovery algorithm, is that this method does not rely on the presence of historical data. When there is a new timetable, we thus don't have to wait until there is enough data to use the causal discovery algorithm to find the graph. Further, some situations might not have happened in the past, so this is not captured in the data and thus not included. The domain knowledge is based on several assumptions, which are described in the following sections.

It is important to define when a train causes a delay on another train. In Figure 7.1, two trains first drive on separate railways, but later they share a section. The orange train is scheduled to use the shared section first, but is delayed. As a result, the blue train has to wait to enter the section and also gets delayed. If we want to determine which train caused the delay of the blue train, two answers may hold. The first option is that the first moment that the orange train is delayed is a direct cause of the delay of the blue train. This is denoted as the edge marked 1 in the figure. To predict the delay of the blue train, the delay of the first delayed orange train is used as input. This is

called a single step prediction. The other option is that the first time the orange train is delayed, resulted in the delay of the next orange trains. Only when the blue train is directly hindered by the orange train, because they need to be at the same railway section, this orange train is the cause of the delay of the blue train. This is denoted as the edge marked 2 in the figure, and this is a multi-step prediction, since multiple predictions need to be made to estimate the delay of the blue train. At the moment of when the prediction is made, only the delay of the first delayed orange train is known.

The second option, the multi-step prediction, is used within this research. Consequently, trains that are not at the same location, cannot have an edge between them in the graph, since they do not have an interaction with each other (yet).

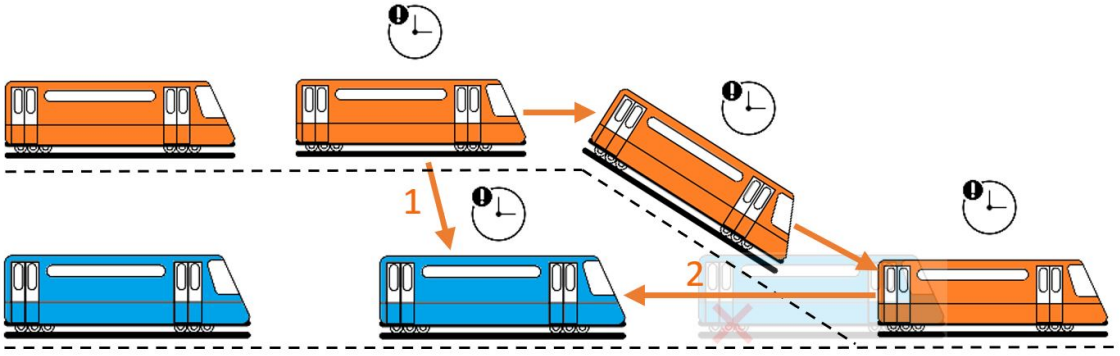

FIGURE 7.1: Two trains, one causing one other a delay

The domain knowledge consists of two main assumptions that always hold.
1. The train is always depending on its previous self, e.g., the same train at successive locations have a dependency between them (**Same train dependency**).
2. If two trains are not within a specified time interval (x) at the same location, we assume that they cannot influence each other's delay, since they are geographically or timewise too far apart from each other.

7.1.1 Definition of 'the same location'

Not all trains are included when using these two previous stated rules, namely the trains that are within x minutes at the same location are left out. The dependency for these trains is not determined yet. This section elaborates on the different options for determining whether two trains can influence each other.

An important decision to make is: what is the definition of 'the same location'? It is clear that a train that is at a certain point in time at a location in the south of the Netherlands will not influence the train that is at that moment in the north of the

Netherlands. Nonetheless, when do we state that a train is at the same location as another train such that they *can* influence each other? In this research, we introduce five different scopes of the location for which trains can interact with each other. The following scopes are ordered by how much train relations it captures and the amount of assumptions that should hold. Although not all scopes may be meaningful, this enumeration includes the whole spectrum of options.

1. **None** There is not any type of location for which trains can influence each other. We only focus on trains that are the same train, but at a different location. This scope only includes the *Same train dependency*, and all the remaining train relations are discarded.

2. **Same platform** Trains that share the same platform within a station within x minutes may influence each other. If a train at this platform is delayed, the next train must wait until the other train has left. This scope also includes the *Same train dependency* assumption.

3. **Same section within a *drp*** In the dataset, the sections are divided per *drp*. If a train shares a section within x minutes with another train, we state that those two trains may influence each other. If a train is delayed and uses a section at a different time, it may cause the other train assigned to that section to wait, as sections can only be used by one train. This scope also includes the assumptions of the *Same platform* scope, since a platform is also a part of a section.

4. **Same station** Trains within the same station may enable a transfer for passengers. If the first train of the transfer is delayed, the other train may wait such that the transfer is still possible. This will only be done if the delay of the second train will not increase too much. The only method to include these transfer dependencies, is to state that every train within the same station within x minutes may influence each other. This scope includes the same assumptions as the *Same section within a drp* scope. The same station is a more general scope than the previous scope, because the sections driven are linked to a certain station. Only if the sections overlap, we include them as a possible edge. Here, any two trains at the station could potentially interact, allowing relationships between trains that enable transfers.

5. **Same drp** Every train that is within x minutes at the same drp could influence each other. This allows for railway interaction between stations and train interaction related to passenger transfers. This scope includes every previous mentioned scope and includes every possible remaining train relation.

7.1.2 Ordering of trains

Another important assumption within this research is that trains do not change their ordering. In reality this happens, but this means that different graphs need to be created, since sometimes the ordering of edges will then be different. Combining these graphs is outside the scope of this research. Therefore, in this research, only one graph is created and the direction of the edges are based on the planned times of the events specified in the schedule.

7.2 Hybrid method

The hybrid method combines the domain knowledge with the PC algorithm. In this method, the term *background knowledge* refers to the domain knowledge, and it contains special properties. The benefit of using background knowledge in our causal discovery algorithm is to enforce that certain edges are included or removed, which consequently reduces the search space.

7.2.1 Background knowledge

The background knowledge consists of two properties: the forbidden edges and the required edges. Section 7.1 describes that there are two assumptions that always hold. These assumptions are included in the background knowledge using the forbidden edges, required edges and possible edges (which are the edges that are not forbidden nor required).

Forbidden edges. There are two situations for which edges are forbidden. Firstly, trains that are at different locations do not cause delays to each other (yet). Secondly, trains that are at the same location, but are time wise too far apart, also cannot cause one another a delay. **Required edges.** There is only one situation for which nodes are marked as required, namely for the same train between successive stops. **Possible edges.** The only edges left are the trains at the same location within a time range that is not too far apart, as described in Section 7.1.1.

In this method, we define the same location as "the same drp". Events that differ in *drp* have by definition forbidden edges. The time interval is chosen to be 15 minutes. Thus, events that are timewise less than 15 minutes apart and are at the same *drp* are the so-called possible edges. These edges are not forbidden or required, so the algorithm must do independence tests to find the dependencies between the edges.

In Figure 7.2, an example with 5 events consisting of 3 trains is shown. The forbidden edges are red, the required edges are green, and the gray dotted edges are the possible edges. The forbidden edges between the trains at station 2 is because the events are time wise too far apart. How to translate the TrainRideObject matrix to background knowledge is explained in Appendix C

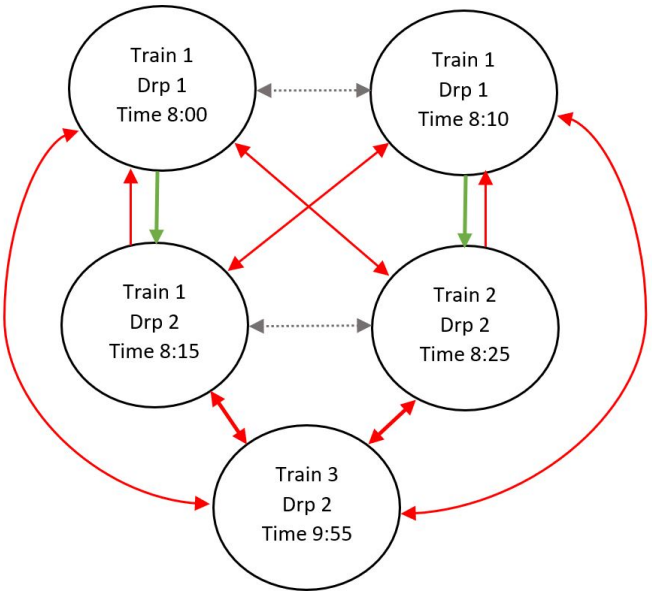

FIGURE 7.2: Forbidden, required and possible edges in red, green, and gray respectively. The same location is defined as "same drp" and the time interval is not explicitly specified, but less than 90 minutes.

7.2.2 Algorithm description

After identifying the background knowledge, the causal discovery algorithm is applied. Initially, the plan was to use a causal discovery algorithm provided by the causal-learn library. However, these algorithms were too slow when working with a large amount of variables, so we decided to study the existing functions and modify the algorithm to improve the computation speed. The main improvements were the adjustment of the BackgroundKnowledge class and the improvement of the Fast Adjacency Search (FAS) algorithm.

Appendix B explains the FAS function of the library, including the usage of the background knowledge. Further, in appendix D, is explained how the BackgroundKnowledge class is improved, by introducing a faster FastBackgroundKnowledge class. Lastly, in appendix E is explained what is done to improve the FAS function.

The causal discovery algorithm consists of the following steps: 1) Start with the complete graph, but exclude the forbidden edges. 2) Conduct a skeleton search on the remaining

edges by performing independence tests. The independence tests between required edges are omitted, since these edges are always kept. 3) Orient the edges. In this section, the steps are elaborated.

7.2.2.1 Start with the complete graph, but exclude the forbidden edges

In this step, we compare each node with the other nodes and check if the two nodes are forbidden. If this is the case, we don't do anything and go to the next node. If they are not forbidden, we mark them as an adjacency by adding them in a dictionary of adjacencies, in both directions.

7.2.2.2 Perform a skeleton search on the remaining edges

In the second step, we perform independence tests between the current adjacencies of each node, to remove other non-causal edges. The Missing Value Fisher-z test is used as independence test with the threshold for statistical significance $\alpha = 0.05$. The working of the independence test is further explained in appendix F. The skeleton search iterates per depth, from 0 to maximally 20, but the function stops earlier if there is no new depth possible. The depth denotes the length of the conditioning set. At depth 0, the conditioning set is empty. The benefit of that the complete graph was first pruned with the forbidden edges, is that the number of adjacencies is reduced significantly. Also, if an edge between two nodes is marked as required, we don't perform an independence test, but directly keep them. Consequently, required edges will never be removed, and since there is no independence test necessary, the algorithm becomes faster.

7.2.3 Orient the edges

When orienting the edges, there is a big difference between the PC-algorithm and the implemented approach. The PC-algorithm orients its edges by identifying the v-structures and using other orientation rules. In this specific application, there are timestamped variables and as stated in section 7.1.2 we assume that the ordering of trains does not change. We therefore assume that the first scheduled train influences the second scheduled train, and not the other way around. This assumption is supported by the definition of ARI, as described in Section 1.2, which states that ARI always (except for some tunnels in The Netherlands) enables the sections according to the planned ordering. So, when a minor disturbance is present, the ordering of actions will not be changed. Concluding, the first event will influence the second event, but not the other way around, and the edges can be oriented based on the timestamps of the nodes.

Chapter 8

Learning assignment functions by implementing Neural Networks

This section provides an overview of the created Neural Networks and which assumptions are made. We elaborate on the assumed noise distribution, how the suitable loss function is calculated and how the different Neural Networks are created and evaluated. Further, we discuss what the layout of each Neural Network is, including which activation functions are used and how many hidden layers are included, and, eventually, which input variables are used.

8.1 Noise distribution

Looking at the data, the noise distribution per *dienstregelpunt* (drp) and train series differs. According to an expert at ProRail, this noise resembles a Gamma distribution. The Gamma distribution differs per *drp*, train series, and whether the train was already delayed at its previous event. To correctly include this noise in the model, there should be two Gamma distributions included per *drp* and train series: one for a delayed train, and one for a non-delayed train.

We decided to assume that the noise contains a Laplace distribution, instead of a Gamma distribution, based on several arguments. 1) The gamma distribution is non-negative, thus negative noise is not allowed. A gamma loss function for a negative value yields infinity, which cannot be processed correctly by the learning algorithm of the Neural Network. Using the Laplace distribution, negative values are allowed. 2) The loss function of a Gamma distribution could be less easy to interpret than the loss function of the Laplace distribution. It is clearer what loss to use for a Laplace distribution (Mean

Absolute Error (MAE), derivation in Appendix H). 3) Such loss function is easier to implement than a derived Gamma loss function, which may differ per drp, train series and initial delay.

The selection of the Laplace distribution instead of another distribution is based on several factors. The first requirement is that the loss function of the distribution should be easy to interpret. The second reason is that the Laplace distribution characterized a double exponential distribution, which looks similar to a (double) Gamma distribution. Thirdly, this distribution allows for negative values. Lastly, the Laplace distribution is less vulnerable to outliers, compared to, for example, the normal distribution. The loss function of a normal distribution is namely the Mean Squared Error (MSE).

8.2 Loss function

A loss function evaluates the performance of a Neural Network by, for example, comparing the predicted outcome to the real outcome. The set of predicted outcomes and the set of real outcomes each have their own set of values. Maximum Likelihood Estimation (MLE) finds the distribution that maximizes the probability of generating data that closely resembles the actual outcomes. The full derivation of the loss function of the Laplace distribution is described in Appendix H. The resulting loss function is the Mean Absolute Error (MAE, $= \frac{1}{m} \sum_{i=1}^{m} |\hat{y}^{(i)} - y^{(i)}|$), which is included in all created Neural Networks.

8.3 Approach for constructing and evaluating the Neural Networks

As stated in Section 8.1, the noise may differ per *dienstregelpunt* (*drp*) and train series. Our approach is to pre-train a Neural Network on the complete train set. Then, the train set is divided in groups per (*drp*, train series), referred to as *buckets*. Per bucket, a Neural Network is trained by fine-tuning the pre-trained NN. Each fine-tuned Neural Network initializes the weights according to the weights calculated by the pre-trained Neural Network. Then, the complete Neural Network is trained with the training set of its own bucket. Each fine-tuned Neural Network is locally saved, such that it can be retrieved for testing purposes. All NN's are Feedforward Neural Networks. A visualization of this approach is denoted in Figure 8.1

To evaluate the model, the test data is separated into buckets of (*drp*, train series). For each bucket, the corresponding Neural Network is retrieved from storage, and evaluated

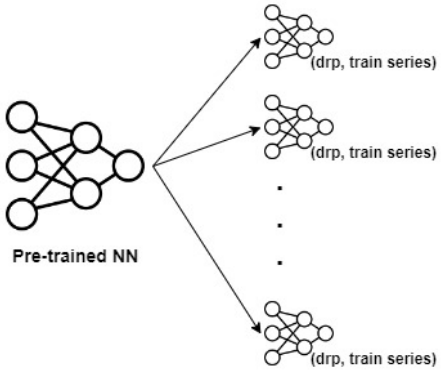

FIGURE 8.1: Visualization of the pre-trained and the fine-tuned Neural Networks

with the bucketed data. A python DataFrame is generated that includes, among other things, the predictions and the actual values. Combining the DataFrames of all fine-tuned Neural Networks, the error metrics are calculated, and the model can be evaluated.

8.4 Layout of the Neural Networks

The pre-trained NN and a fine-tuned NN consist of the same structure. It consists of an input layer, a dense hidden layer with 256 neurons and an output layer of 1, which denotes the delay in seconds (Figure 8.2). The number of neurons in the hidden layer is determined by testing and evaluating a subset of possibilities. The activation function in the hidden layer is ReLu and the activation function in the output layer is PReLU (Parametric ReLu) (Figure 8.3). These functions are chosen by performing a greedy search, by testing all possible combinations of activation functions. An important aspect of a PReLu activation function is that the output can be negative as well. In our dataset, negative delays also occur.

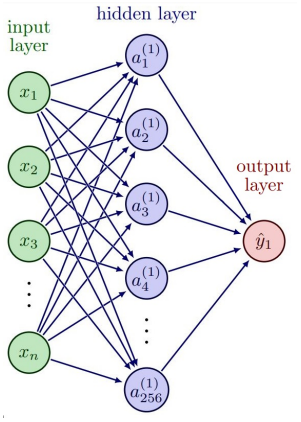

FIGURE 8.2: NN layout

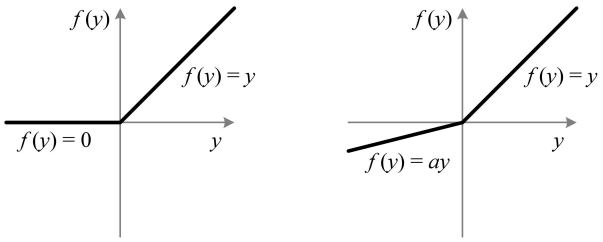

FIGURE 8.3: Left: ReLu, Right: PReLu

8.5 Input variables

The input variables consist of 21 input features and can be divided in 3 categories: 1) information that is extracted from the found causal graph, 2) time-table-related variables and 3) infrastructure- or train material related variables. The exact method how the input variables are extracted from the causal graph is described in Appendix G.

1) Input features extracted from the found causal graph:

prev_event This denotes the delay of the same train at the previous event

prev_prev_event This denotes the delay of the same train at the 2nd previous event

dep0_platform The delay of the train that was before this train at the same platform. If this train is not included, this delay is default -50

dep1 The delay of a possible interacting train that is **not** at the same platform. The trains that are included in dep1 to dep3 are ordered by their planned time: the first train is denoted in dep1, the second train in dep 2 and the third train is in dep 3. If there is no possible interacting train to include, the value of the column is by default -50.

dep2 See explanation at dep1.

dep3 [1] See explanation at dep1.

time interval0 Time difference between the planned time of this train and the train at dep0_platform.

time interval1 Time difference between the planned time of this train and the train at dep1.

time interval2 Time difference between the planned time of this train and the train at dep2.

time interval3 Time difference between the planned time of this train and the train at dep3.

[1] The algorithm dynamically chooses the number of dependencies for a given dataset, based on the maximal number of non-trivial parents that the nodes in the causal graph contains. In our research, 4 non-trivial parents (including dep0_platform) was the max

2) Timetable-related variables:

buffer This value is only filled for the events with a departing activity (K_V or V), that denotes the difference of planned time in seconds between arrival and departing at that station.

travel time This denotes difference in time between two consecutive events of the same train. It is calculated by: *planned time — the planned time of the previous event.*

activity This variable denotes a one-hot encoding of the activity of the train (V, A, K_V, K_A, and D).

minimal dwell time For some stations, there is a minimal time that a train should stay at the station. This minimal dwell time also differs per train type.

slack Slack is the planned travel time *minus* the strictly necessary travel time. The strictly necessary travel time is a given column in the data set.

day of the week This variable is a one-hot encoding that denotes one of the 5 days of the workweek.

peakhour This variable is a one-hot-encoding that denotes which type of time the activity occurs. There are 3 options: peak hour in the morning, non-peak hour and peak hour in the evening.

3) Infrastructure- or train material related variables:

travel distance This denotes the travel distance between two *drp* in hectometers

speed This denotes the allowed speed on this section

type_station This denotes a one-hot encoding regarding the types of stations. The types of stations are specified in an external file and extracted into python.

traintype This variable denotes which type of train it is (Sprinter, Intercity, LM (empty train), HSN (High speed train) or Thalys).

Chapter 9

Results

In this section, the results of the created model are discussed. The first section provides an evaluation of the created causal graph. The second section provides a description of the created Neural Networks and the train-, test-, and validation data. In the third section, our model is evaluated on the complete test set by looking at the results per station. In Section 9.4, the model is evaluated by only focussing on a test set for which there is known that there was an interaction. In addition, Section 9.5 compares our model to another model to estimate the negative effects are when our model misses (partial) information about the possible non-trivial parents. Then, in Section 9.6, the benefit of fine-tuning the model is described. Subsequently, in Section 9.7, the results of our model are compared to the results of an existing model. Lastly, Section 9.8 provides a conclusion based on the results.

This chapter focuses on the results of the Rotterdam-Dordrecht (Rtd-Ddr) line. To limit the number of pages in the Result section, the results for the Amsterdam-Utrecht and the Utrecht-Eindhoven line are denoted in Appendix I and J.

9.1 Evaluation of the causal graph

The hybrid algorithm described in Section 7.2 yields a causal graph as a result. When applying this algorithm to the dataset of the Rtd-Ddr line, a large graph is constructed. A part of the graph is illustrated in Figure 9.1. The graph resembles a large tree, where delays could propagate downwards. The majority of edges are present between the same train at consecutive events. In some cases, there is a relation identified between different trains at the same *drp*.

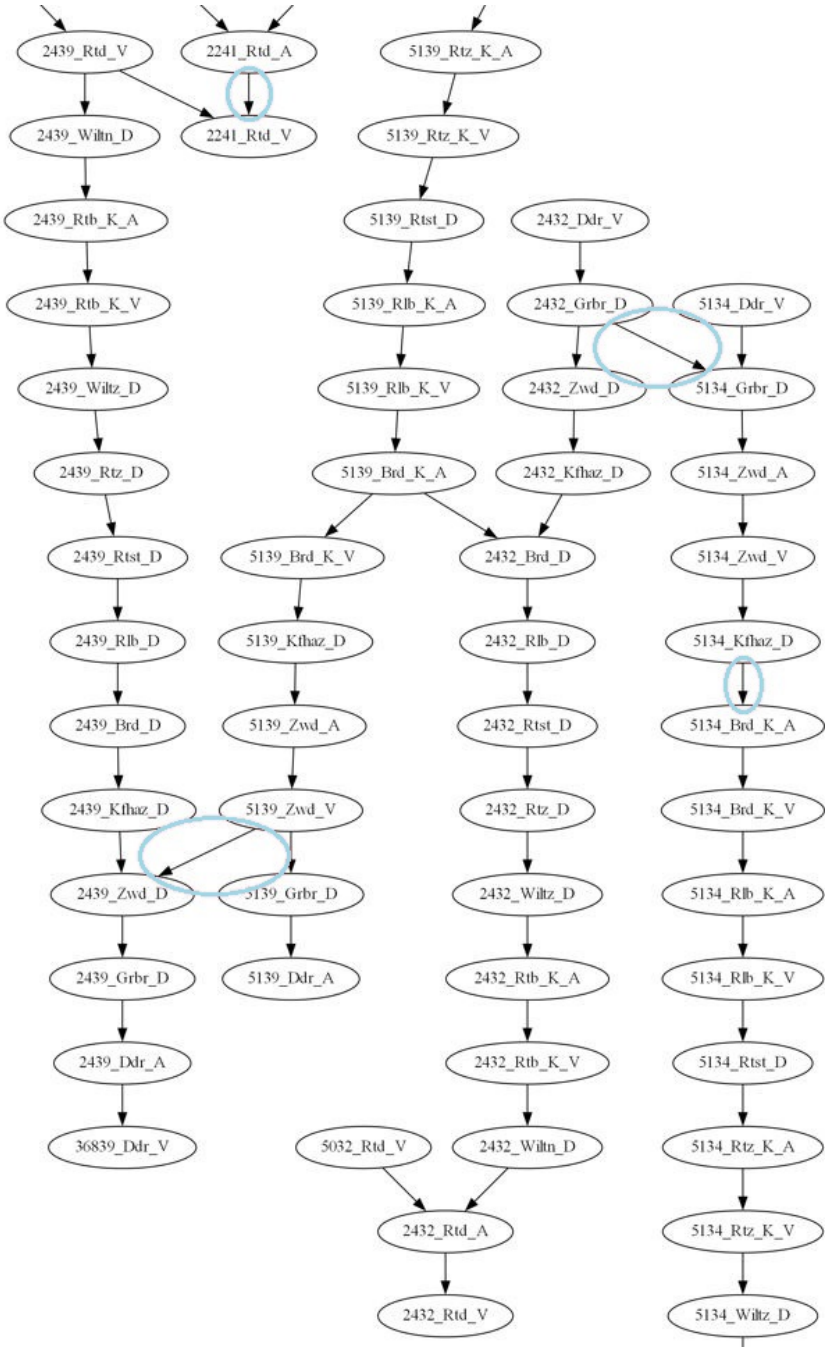

FIGURE 9.1: Visualization of the partial causal graph of the Rotterdam-Dordrecht line. The four blue circled edges are further analyzed in this research.

The four edges marked with a blue circle in this graph are further analyzed. The correlation between the same train at two consecutive *drp*s are shown in Figures 9.2 and 9.3. In cases where there is a buffer present between two consecutive events, namely arriving and departing, a distinct pattern can be observed. First, there is a short segment with no evident correlation, then followed by a linear correlation. This is shown in Figure 9.3, where the linearity starts at around 180 seconds (3 minutes) This is also the size of the buffer.

Further, the relations between different trains are analyzed. In Figure 9.4, both trains have to arrive at or pass through the same platform. After at least 435 seconds (more than 7 minutes), the delay of train 2439 is not in proportion to the delay of train 5139 as anticipated. The reason for this is that the trains most likely were switched in ordering. This is not defined in the TAD (Train Handling Document), but it is probable that a train dispatcher intervened.

Figure 9.5 depicts two trains that share multiple sections at *drp* Grbr (Grote Brug). The correlation is less visible than the other three figures, since the 2432 hindered the 5132 six times that resulted in a small delay.

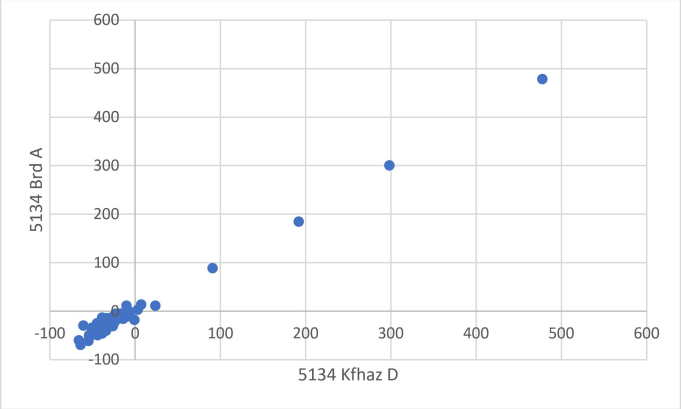

FIGURE 9.2: Correlation between the same train 5134 between *drp*s Kfhaz and Brd

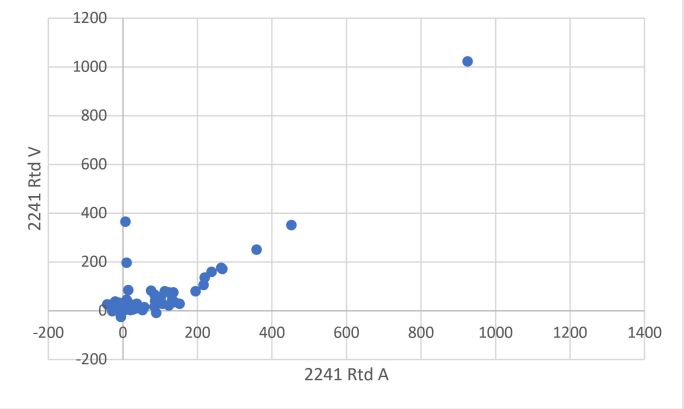

FIGURE 9.3: Correlation between the same train 2241 between at the same station Rtd, arriving and departing. These events have a buffer of 3 minutes, which is noticeable in the graph

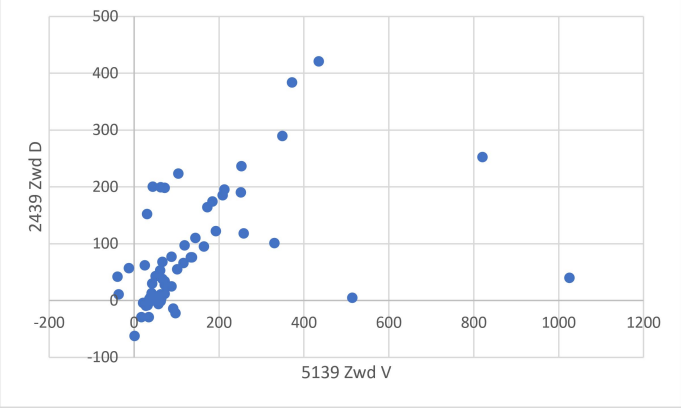

FIGURE 9.4: Correlation between two trains that use the same platform. After more than 435 seconds (around 7 minutes) the ordering of trains is switched, such that train 2439 will not be delayed by train 5139

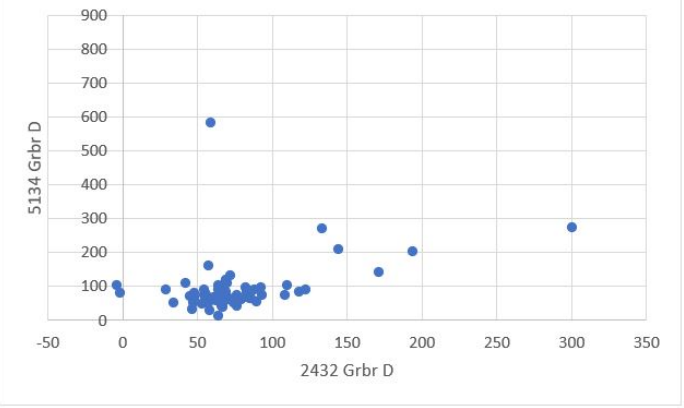

FIGURE 9.5: These trains do not share a platform, but do pass through the same station and share multiple sections. The correlation is less visible than the other three figures, since the 2432 hindered the 5132 six times that resulted in a small delay

9.2 Number of Neural Networks and number of train-, test-, and validation data

The pre-trained model for the Rtd-Ddr section is trained with 324,131 training samples, 64,826 validation samples and tested with 81,033 samples. The fine-tuned model consists of 252 Neural Networks. The maximum number of samples to train the NN per bucket is 4459 samples and the minimum number of samples is 16. The mean number of samples per bucket is 1286, and the median number is 868.

9.3 Model evaluation on the total test set and per station

The constructed model is evaluated using two metrics, the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). The equations are as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y} - y|$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2}$$

For the total test set, the MAE is 9.12 seconds and the RMSE is 32.44 seconds. The average prediction error in terms of MAE and RMSE per *drp* are shown in Figures 9.6 and 9.7. The largest MAEs, observed at Rtd and Ddr, can be explained by the fact that both *drp*s are large stations consisting of several platforms, which allows for more interaction. The spike of the RMSE value at *drp* Rtst is due to the fact that there are 2 large estimation errors (2157 and 1266 seconds too low) which influences the results largely. Removing those two predictions, the MAE and RMSE of Rtst drop to 5.77 and 21.66 respectively.

9.4 Testing our model with more specific test sets

In this section, the results are further analyzed by focussing on the events for which it is known that a train interaction took place. When ProRail automatically registered a train interaction, the column `cause' is filled with the corresponding train number. A downside to this column is that if a train only hindered another train for even one second, this train will be included in the column. Such interactions are too small to be noticed by our algorithm. The dataset is therefore further reduced by removing rows

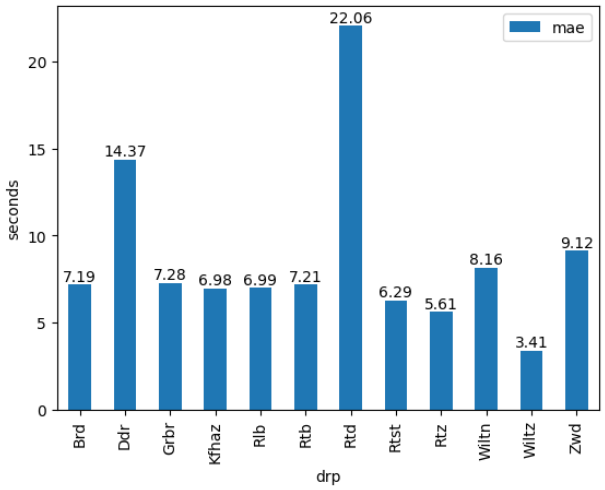

FIGURE 9.6: MAE per drp

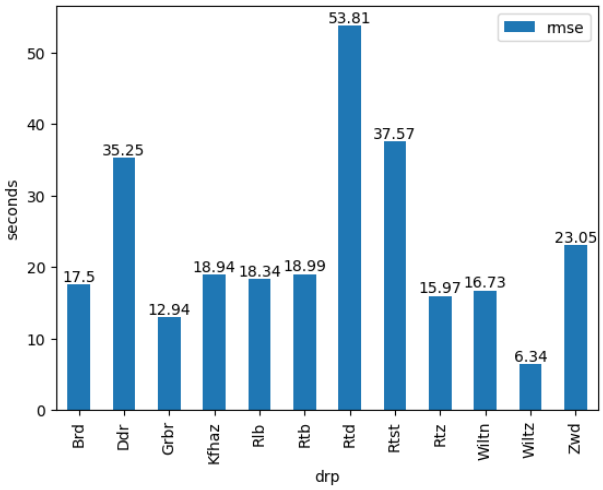

FIGURE 9.7: RMSE per drp

where our algorithm did not observe a possible interaction of non-trivial parents (i.e.: dep0 to dep3 are empty). This subset is called: subset of rows including non-trivial parents and where ProRail found an interaction.

In order to test the effect of the non-trivial parents included in our model, our model is tested against another model. Our model is trained and tested with data containing the columns about non-trivial parents. The other model is trained and tested on the same sets, but the columns about the non-trivial parents are not present. In this chapter, we will refer to the model with the columns that includes the non-trivial parents as `*our model*' and the other model without the columns regarding non-trivial parents as the `*baseline model*'. The results are denoted in Table 9.1.

Test-set	**MAE**	**RMSE**	**# of test-samples**
Our model: Total test set	9.12	32.44	81,033
Baseline model: Total test	9.06	32.45	81,033
Our model: subset of rows including where ProRail found an interaction	44.47	74.24	3201
Baseline model: subset of rows including where ProRail found an interaction	44.94	75.53	3201
Our model: subset of rows including non-trivial parents	17.95	57.67	2294
Baseline model: subset of rows including non-trivial parents	18.06	59.06	2294
Our model: subset of rows including non-trivial parents and where ProRail found an interaction	54.83	80.83	215
Baseline model: subset of rows including non-trivial parents and where ProRail found an interaction	69.33	100.40	215

TABLE 9.1: Results of the Rtd-Ddr line from our model and the baseline model

9.5 Classifying the predictions of our model by means of the baseline model

The goal of this section is to estimate what the negative effects are when our model misses (partial) information about the possible non-trivial parents. The last two rows of Table 9.1, namely, only shows the best subset to test on: when our model predicts an interaction and ProRail confirmed these interactions. These are the so called True-Positive cases. The other cases (True-Negative, False-Positive, and False-Negative) are evaluated in this section according to the following approach.

The two models, our model and the baseline model, described in Section 9.4 are tested on the complete test set. The predictions of our model are compared with the predictions of the baseline model, and the samples where the prediction differs more than 15 seconds are kept to be analyzed further. We group the samples by if there was a non-trivial parent found in the causal graph for that sample, and which model has a better prediction on that sample. In total, we get 4 groups: our model performed best and there was a non-trivial parent found in the graph, our model performed best and there was **not** a non-trivial parent found in the graph, the baseline model performed best and there was a non-trivial parent found in the graph, and the baseline model performed best and there was **not** a non-trivial parent found in the graph. Each group denotes a classification class, as shown in Table 9.2. Each cell is elaborated in the list provided below the table.

Neural Network that performed best	**subset where non-trivial parents were found in the causal graph**	**subset where non-trivial parents were not found in the causal graph**
Our model performed best	True Positive	True Negative
The baseline model performed best	False Positive	False Negative

TABLE 9.2: Denoting per situation to which classification class it belongs to

True Positive These samples were better predicted by our model than by the baseline model. Including the known non-trivial parents improved the prediction of our model, thus the information about the non-trivial parents was correct

True Negative These samples were better predicted by our model than by the baseline model. Including the information that there are no non-trivial parents present was correct, since it benefitted the predictions for our model. Namely, this information states that there are no non-trivial parents, which means that our model can anticipate on this.

False Positive These samples were better predicted by the baseline model than by our model. Our model received information about that there would be a non-trivial parent, but this worsened the predictions, since the baseline model predicted better. An explanation would be that the assignment functions in our model are learned incorrectly, because there were too little number of samples, and thus the model is unable to process this information correctly. (Therefore, this would not be per definition a False Positive, but for consistency, the cell is named False Positive)

False Negative These samples were better predicted by the baseline model than by our model. Our model received information about that there are no non-trivial parents, but this worsened the predictions and the baseline model predicted better. An explanation would be that there are non-trivial parents present in reality, but were missed when creating the causal graph.

Our model and the baseline model were tested accordingly with the dataset of the Rotterdam-Dordrecht line. There are in total 456 samples where our model and the baseline model differ in their predictions by at least 15 seconds. In 236/456 cases (52%), our model made a better prediction than the baseline model. From those 236 cases, 105 cases (44%) included information about a non-trivial parents. The other 131 cases did not have non-trivial parents. The results are denoted in Table 9.3.

Neural Network that performed best	**subset where non-trivial parents were found in the causal graph**	**subset where non-trivial parents were not found in the causal graph**	**Total**
Our model performed best	105	131	**236**
The baseline model performed best	90	130	**220**
Total	**195**	**261**	**456**

TABLE 9.3: Comparing our model with the baseline model, in terms of how often each model performed better.

To quantify what the (mean) differences in estimations are between per model per cell, Table 9.4 is created, where the differences in terms of MAE are denoted. In the top left cell, "68.42 vs 112.03" means that our model had a score of 68.42 seconds and the baseline model had a score of 112.03 seconds. The mean difference between the baseline model and our model for the top left cell was 43.61 seconds.

Comparing the results in the column `subset where non-trivial parents were found in the causal graph', the mean difference in error per row is approximately 43 seconds (43.61 and 42.86). This means that including non-trivial parents in the Neural Network is as much benefitting as worsening the predictions, so it is important to identify the

Neural Network	subset where non-trivial parents were found in the causal graph	subset where non-trivial parents were not found in the causal graph
Our model performed best	68.42 vs 112.03 (decrease of 43.61)	37.05 vs 59.68 (decrease of 22.63)
The baseline model performed best	83.11 vs 40.25 (increase of 42.86)	73.42 vs 47.49 (increase of 25.93)

TABLE 9.4: Comparing our model and the baseline model in terms of MAE in seconds. The scores are denoted as follows: MAE score of our model vs MAE score of the baseline model. The difference is determined by: MAE score of our model – MAE score of the baseline model.

non-trivial parents well. Comparing the results in the column `subset where non-trivial parents were not found in the causal graph', both mean differences lie around 24.5 seconds. This means that the advantage of identifying the absence of non-trivial parents correctly is equivalent to the disadvantage when identifying this falsely.

9.6 Impact of the fine-tuned model

As described in Section 8.3, a fine-tuned model is created by grouping the different sets of (*drp*, train series) and per group, a specific Neural Network is trained further, containing the pre-trained Neural Network as the initial weights. This section researches the effect of the fine-tuned model, by comparing it to the pre-trained model in terms of MAE and RMSE. As shown in Table 9.5, the fine-tuned model outperforms the pre-trained model. The difference in error gets bigger when focussing more on the test set that contains a larger ratio of interacting trains. This means that fine-tuning the Neural Networks improves the predictions for trains with interactions.

Model and subset	MAE	RMSE	Number of test-samples
Pre-trained tested on total dataset	10.30	33.75	81,033
Fine-tuned tested on total dataset	9.12	32.44	81,033
Pre-trained tested with subset ProRails interaction registration	49.20	78.78	3201
Fine-tuned tested with subset ProRails interaction registration	44.47	74.24	3201
Pre-trained tested with subset of rows including non-trivial parents and where ProRail found an interaction	69.16	96.99	215
Fine-tuned tested with subset of rows including non-trivial parents and where ProRail found an interaction	54.83	80.83	215

TABLE 9.5: Comparing the results of the pre-trained model to the results of the fine-tuned model, tested with three different test tests

9.7 Comparing the result to the paper of Wen, Mou, et al. (2020)

Wen, Mou, et al. (2020) used a LSTM prediction model to predict the arrival delays on the section Rtd-Ddr on the Dutch railway line. They also used the dataset of historical data ranging from September 4, 2017, to December 8, 2017, excluding weekends. There are three main differences between their research and our research: 1) Their research only uses the intermediate stations, instead of all *drp*s. 2) They only predicted the delay of arriving trains, and not departing trains or trains passing through. 3) In their paper it is not defined how the data is preprocessed, but according to their plotted delay distribution (Figure 9.8), the delays lie between -1000 and 2200 seconds. This is a different delay distribution than our generated test set (Figure 9.9).

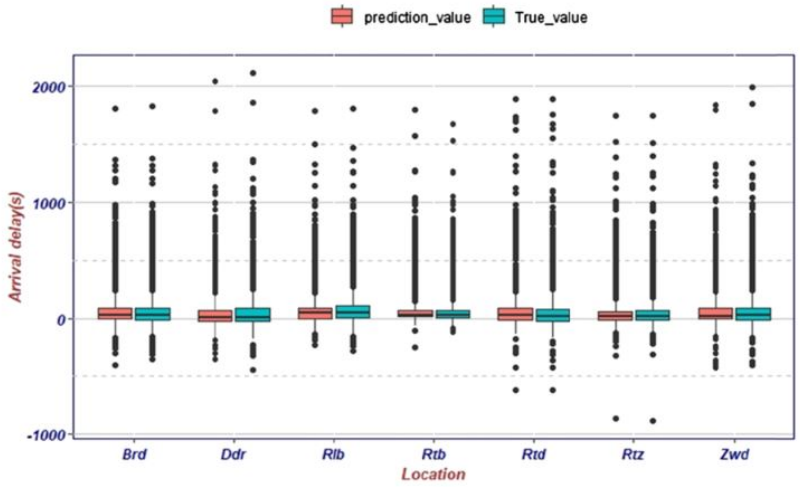

FIGURE 9.8: Distribution of the delay data used in the paper of Wen, Mou, et al. (2020)

To compare their method to ours in this section, we have adjusted our dataset and method to theirs. The following three modifications are made to our dataset. 1) The dataset is altered by only keeping the same stations as mentioned in the paper. 2) For our prediction, the information about passage or departure is kept (this was also done in the paper), but in our evaluation, only the arrival predictions are used. 3) We trained the model with the outliers, but in our evaluation, we only evaluate the predictions that have a true value ranging between [-1000, 2200]. The distribution used in our prediction is shown in Figure 9.9.

Comparing the results regarding MAE of the model (Figure 9.10), our model predicts 4 of the 7 cases better than the paper, with a difference of at least 3.85 seconds. In the other cases, the difference in error between the predictions per *drp* is maximally 0.97 seconds.

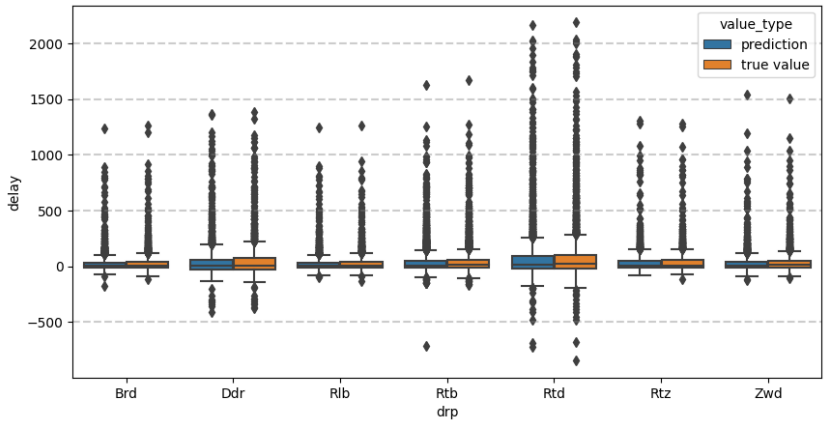

FIGURE 9.9: Distribution of the delay data used in our research

The exact errors regarding the RMSE in the paper are not shown [Wen, Mou, et al. 2020]. However, Mou et al. (2019) included the same results in their paper by showing a graph without the exact numbers. The numbers are approximated and included in Figure 9.11. Also in this case, 4 out of 7 cases are better predicted by our model than the model from the paper; However, it is a small difference. Comparing the 3 results that have fallen behind, those differences are bigger. Given that the distributions are not exactly the same, we cannot state that one method outperforms the other. However, it can be stated that the results are quite similar to each other, indicating that our model performs comparably to the other model.

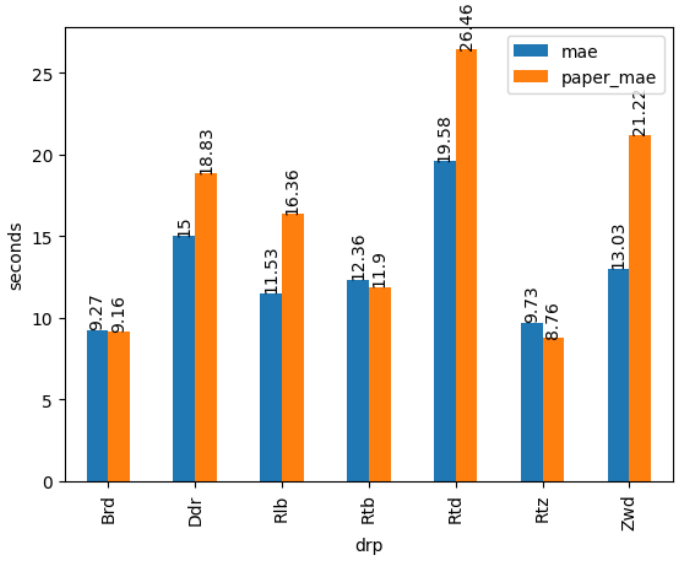

FIGURE 9.10: MAE: Comparing our results (blue) with the results described in the paper of Wen, Mou, et al. (2020) (orange)

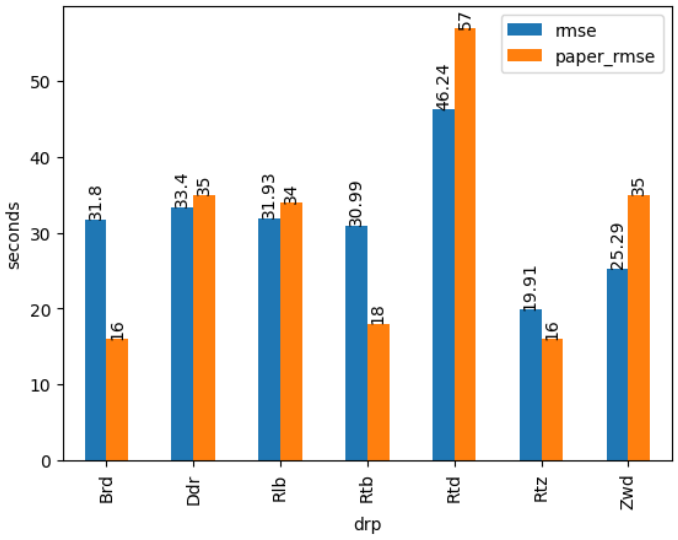

FIGURE 9.11: RMSE: Comparing our results (blue) with the results described in the paper of Mou et al. (2019) (orange)

9.8 Conclusion

The outcomes of this research show that finding and implementing causal relations between interacting trains in a prediction model improves the delay predictions. When only looking at the cases where there was in reality an interaction present and this was also identified by the causal discovery algorithm, the average prediction error is significantly lower than the model that is not trained with possible train interactions.

In these cases, the prediction error on the next stop is improved for the different datasets Rtd-Ddr, Asd-Ut, and Ut-Ehv with 14.5, 6.81, and 18.44 seconds respectively. If we extend this method to predict the delays across multiple stops, the improvement in error would be increasing. Starting at the first station and predicting the delay at the last station, the improvement in errors would be as followed: from Rotterdam to Dordrecht (12 timetabling points), the difference in error would be $14.5 \cdot 12 = 3$ minutes. For Amsterdam - Utrecht (18 timetabling points): $6.82 \cdot 18 = 2$ minutes, and for Utrecht - Eindhoven (23 timetabling points): $18.44 \cdot 23 = 7$ minutes.

However, sometimes including possible train interaction could worsen the predictions if the assignment functions are not accurately learned. Insufficient number of samples containing train interactions to train on could lead to this situation. Also, when possible train interactions are mistakenly not found by our causal discovery algorithm, our prediction is worse than the model that is not trained with possible train interactions.

Additionally, a comparison is made between the model trained with a single neural network and the model consisting of several fine-tuned neural networks. It is observed that fine-tuning the model is beneficial for the delay predictions. Lastly, the results obtained from our model are compared to those of an existing model, demonstrating a similar performance.

Chapter 10

Discussion and Conclusion

The goal of this thesis was to detect causal relationships between interacting trains and include this information in our prediction model. The research question was:

> How to derive a causal graph from train network data in the presence of background knowledge and learn the assignment functions?

To answer this main research question, 4 sub questions were created. In the following paragraphs, the most important answers to these questions are provided. Further, the future work is discussed and lastly, the main research question is answered.

Subquestions:

- What does the train network data look like?
- Which methods can be used to create a SCM graph for the train network data?
- How can the assignment functions be learned using Neural Networks?
- How can the created SCM's be evaluated?

The most important aspect of the train network data is that not every day the same events occur. It is therefore important to delete and insert certain events such that each day consists of the same events. Further, according to an expert at ProRail, the noise distribution in our data would be a Gamma distribution, but in our research this is approximated to a Laplace distribution.

To create a causal graph, the causal-learn library was used to implement the causal algorithm. The PC-algorithm in combination with background knowledge is applied in

our research. The background knowledge was added to enforce the inclusion or exclusion of edges and to prune the search space. The implementation of the library was very slow, so the algorithm is modified such that the computation time dropped significantly.

The assignment functions are learned by first training a pre-trained Neural Network on the complete training set. Then, the training set is grouped into buckets of (*drp*, train serie), and per bucket, the pre-trained Neural Network is further trained, resulting in a set of fine-tuned Neural Networks. Conclusively, our model consists of a set of several fine-tuned Neural Networks.

The created SCM is evaluated on two aspects, the causal graph and the model consisting of the fine-tuned Neural Network. The created causal graph is evaluated by selectively examining specific edges to find which kind of relation it describes (e.g., are they the same train, do they share a section or is it a transfer). Also, the delays of the nodes of some edges are plotted against each other. Secondly, the complete model is evaluated by comparing the model to a baseline model and an existing model in terms of MAE and RMSE.

As indicated in Section 9.8, our causal discovery algorithm may not identify all train interactions, or there may be inaccuracies in the learned assignment functions. Sometimes a train interaction has only occurred once in the historical dataset, which would not be significant when finding causal relations. Further, learning the correct assignment functions based on a very small amount of train interactions is not possible. However, when the possible train interactions are found and the assignment functions are calculated correctly, our model improves significantly.

Furthermore, our prediction model does not include freight trains or machinist transfers. As a result, our model has to work with a certain level of noise, which may introduce inaccuracies in some predictions.

10.1 Future work

This section is split up in 3 parts; The first section elaborates on future work regarding finding the causal graph, consisting of four items. The second section elaborates on future work regarding finding the assignment functions, consisting of two items. The last section consists of future work that are more general, consisting of four items.

10.1.1 Future work regarding finding the causal graph

In this research, the constraint based PC-algorithm is applied. It would also be interesting to apply a score based algorithm to extract the causal graph, and research what the

difference in result would be. An approach for the Greedy Equivalent Score would be to skip the forward stage and start with the backward stage, where a initial graph skeleton is already created using background knowledge. This skeleton would then already have the forbidden edges pruned. When pruning the initial graph further, the included background knowledge can prevent that required edges will be removed. One difficulty to this algorithm would be to find a suitable score function, since our noise differs per *drp*, train series and if the train was initially delayed.

Further, the Fast Adjacency Search step of the PC algorithm contains a threshold α for the significance level of the independence tests. In our research, this threshold is set to $\alpha = 0.05$. Since our dataset consists of a limited number of train interactions, this threshold might be too low to find the interactions that occurred only a few times. A possible solution could be to decrease this α-value. The benefit of this is that more correct relations that did not occur many times in our dataset would be captured.

Moreover, the causal graph is evaluated by electively examining specific edges and checking if these were logical. For future work, the causal graph could be evaluated by using an evaluation tool for the causal graph.

Lastly, we made the assumption that the ordering of trains does not change. We assume that the ordering of trains is according to the predefined schedule. In reality, changes in ordering of trains do occur. In a further research, it would be interesting to investigate how the change of relations between trains can be included in the model.

10.1.2 Future work regarding finding assignment functions

To find the suitable loss function, we made the assumption that the noise distribution of the delay variables would be Laplace. In reality, it is a Gamma distribution. In a further research, we would recommend finding a suitable loss function(s) for a Gamma distribution. Since the Gamma distribution differs per train series, drp and if they are delayed, it is possible that multiple loss functions need to be created.

Further, the input for the Neural Network consists of, among other things, the included non-trivial parents (dep0 to dep3). The column dep0 is assigned when the other interacting train is at the same platform. The other dep1 to dep3 are ordered by the planned times of the interacting trains. The disadvantage is as follows: the buckets to train the NN on are grouped per train serie, which means that it combines several train numbers and their non-trivial parents. One train number may contain another train series as non-trivial parent than another train number in that bucket. An example is shown in Figure 10.1. Here, there are two trains that both only have one non-trivial

parent, each of another train serie. In the current model, both delays are stored in the same column. In reality, the assignment functions between these two relations would be different, and our model has to approximate the function to satisfy both relations. A improvement would be to not divide the columns in dep1-dep3, but on the train series, as described in Figure 10.2. Here, within each column, the assignment functions would be approximately the same.

Train number	Dep1
5024	Delay of train 7424 (train series 7400)
5026	Delay of train 4026 (train series 4000)

FIGURE 10.1: Current method for including dependencies. If there is only one non-trivial parent that is not on the same platform, it will automatically be placed in dep1

Train number	Delays of train series 7400	Delays of train series 4000
5024	Delay of train 7424	
5026		Delay of train 4026

FIGURE 10.2: Proposed method for including non-trivial parents. Each non-trivial parent is divided in the column of their own train series

10.1.3 Future work in general

This research focused on predicting the delay on the next *drp*/timetabling point of a train. However, our research goal is the exploration of causal analysis applied to delay data, which could be a precursor to a decision support system. Such decision support system for the train dispatchers must be able to predict the delays several stops ahead. It would be interesting to perform research on implementing a system that extends our model such that it would be able to predict ahead of several timetabling points. Using this system, it is also possible to assess the performance of our model in predicting several future stops.

Another goal of the decision support system is to provide suggestions on when and how to intervene between trains. Additionally, not only a decision support system makes such decisions, we also should predict which decisions a train dispatcher would make. When knowing their decisions, the decision support system could anticipate on this. For future work, one could research how to use the learned SCM to make decisions regarding intervening between trains.

Another potential area of future work is related to the expansion of the prediction model. Our research focussed on finding train interaction to include in our model. This left out elements such as personnel and freight trains. For a future research, it could be

researched how to include this information as well, to create a more accurate prediction system.

Also, our research is based on historical data. Only the events that occurred within the given dataset are captured. However, not all events that *could* happen would be included in our model and, thus, other train interactions could be missed. Also, when a train interaction *is* captured by our causal discovery model, there might be an insufficient number of events to correctly learn the assignment functions from. For our exploration of causal analysis applied to delay data, it would be interesting to have included more train interaction in our dataset, since this is not a large percentage of our historical data. A solution could be to use a simulation program of ProRail to simulate situations where there are disturbances, such that our dataset contains more situations where there are train interactions.

10.2 Concluding remark

The main research question was *How to derive a causal graph from train network data in the presence of background knowledge and learn the assignment functions?*. To achieve this result, the research was split up in two parts: finding the causal graph and learning the assignment functions. To find the causal graph, a hybrid algorithm is applied that combines the created background knowledge with the PC-algorithm. To learn the assignment functions, multiple Neural Networks are trained and tested. Our model is tested against a version that does not take possible train interactions into account (the baseline model) and against an existing method.

Comparing the results, it can be concluded that conducting causal analysis on delay data shows promise for further research. The method to include causal relations between trains could be incorporated by ProRail as an extension to a delay prediction method. However, as stated in the previous section, several enhancements are required to make this approach more suitable for delay prediction.

Bibliography

Causal-learn (2022a). *BackgroundKnowledge.py*. URL: https://github.com/py-why/causal-learn/blob/0.1.3.0/causallearn/utils/PCUtils/BackgroundKnowledge.py. (tag: 0.1.3.0).

– (2022b). *FCI.py*. URL: https://github.com/py-why/causal-learn/blob/0.1.3.0/causallearn/search/ConstraintBased/FCI.py. (tag: 0.1.3.0).

– (2022c). *PC.py*. URL: https://github.com/py-why/causal-learn/blob/0.1.3.0/causallearn/search/ConstraintBased/PC.py. (tag: 0.1.3.0).

Chickering, David Maxwell (2002). "Optimal structure identification with greedy search". In: *Journal of machine learning research* 3.Nov, pp. 507–554.

CLeaR (2022). *Causal learn library, cit.py, line 167*. URL: https://github.com/cmu-phil/causal-learn/blob/5eaf0f606476780ffef4dd56e2a8a7741853bd1b/causallearn/utils/cit.py#L167. (accessed: 26-01-2023).

Corman, Francesco and Pavle Kecman (2018). "Stochastic prediction of train delays in real-time using Bayesian networks". In: *Transportation Research Part C: Emerging Technologies* 95, pp. 599–615.

Dekker, Mark M et al. (2021). "A next step in disruption management: Combining operations research and complexity science". In: *Public Transport*, pp. 1–22.

Glen, Stephanie (n.d.). *Fisher Z-transformation*. URL: https://www.statisticshowto.com/fisher-z/. (accessed: 26-04-2023).

Glymour, Clark, Kun Zhang, and Peter Spirtes (2019). "Review of causal discovery methods based on graphical models". In: *Frontiers in genetics* 10, p. 524.

Huang, Biwei, Kun Zhang, et al. (2018). "Generalized score functions for causal discovery". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1551–1560.

Huang, Ping, Javad Lessan, et al. (2020). "A Bayesian network model to predict the effects of interruptions on train operations". In: *Transportation Research Part C: Emerging Technologies* 114, pp. 338–358.

Huang, Ping, Thomas Spanninger, and Francesco Corman (2022). "Enhancing the understanding of train delays with delay evolution pattern discovery: A clustering and

Bayesian network approach". In: *IEEE Transactions on Intelligent Transportation Systems* 23.9, pp. 15367–15381.

Huang, Ping, Chao Wen, Liping Fu, Javad Lessan, et al. (2020). "Modeling train operation as sequences: A study of delay prediction with operation and weather data". In: *Transportation research part E: logistics and transportation review* 141, p. 102022.

Huang, Ping, Chao Wen, Liping Fu, Qiyuan Peng, et al. (2020). "A deep learning approach for multi-attribute data: A study of train delay prediction in railway systems". In: *Information Sciences* 516, pp. 234–253.

Koller, Daphne and Nir Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Lessan, Javad, Liping Fu, and Chao Wen (2019). "A hybrid Bayesian network model for predicting delays in train operations". In: *Computers & Industrial Engineering* 127, pp. 1214–1222.

Li, Dewei, Winnie Daamen, and Rob MP Goverde (2016). "Estimation of train dwell time at short stops based on track occupation event data: A study at a Dutch railway station". In: *Journal of Advanced Transportation* 50.5, pp. 877–896.

Li, Dewei, Rob MP Goverde, et al. (2014). "Train dwell time distributions at short stop stations". In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 2410–2415.

Li, Zhongcan, Ping Huang, et al. (2022). "Prediction of train arrival delays considering route conflicts at multi-line stations". In: *Transportation Research Part C: Emerging Technologies* 138, p. 103606.

Luo, Jie, Ping Huang, and Qiyuan Peng (2022). "A multi-output deep learning model based on Bayesian optimization for sequential train delays prediction". In: *International Journal of Rail Transportation*, pp. 1–27.

Meek, Christopher (2013). "Causal inference and causal explanation with background knowledge". In: *arXiv preprint arXiv:1302.4972*.

Mou, Weiwei, Zhaolan Cheng, and Chao Wen (2019). "Predictive model of train delays in a railway system". In: *RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019*. 069. Linköping University Electronic Press, pp. 913–929.

Movares and Prorail (2022). "Telefoonnummers van Verkeersleidinggebieden in Nederland". In: (version: 9.2 Vervoerders).

Murali, Pavankumar et al. (2010). "A delay estimation technique for single and double-track railroads". In: *Transportation Research Part E: Logistics and Transportation Review* 46.4, pp. 483–495.

Nandy, Preetam, Alain Hauser, and Marloes H Maathuis (2018). "High-dimensional consistency in score-based and hybrid structure learning". In: *The Annals of Statistics* 46.6A, pp. 3151–3183.

Nielsen, Lars Kjær, Leo Kroon, and Gábor Maróti (2012). "A rolling horizon approach for disruption management of railway rolling stock". In: *European Journal of Operational Research* 220.2, pp. 496–509.

NOS (2022). *Weer minder treinen vanwege personeelstekort bij verkeersleiding ProRail*. URL: https://nos.nl/artikel/2416046-weer-minder-treinen-vanwege-personeelstekort-bij-verkeersleiding-prorail. (accessed: 01-02-2023).

Ogarrio, Juan Miguel, Peter Spirtes, and Joe Ramsey (2016). "A hybrid causal search algorithm for latent variable models". In: *Conference on probabilistic graphical models*. PMLR, pp. 368–379.

Oneto, Luca et al. (2018). "Train delay prediction systems: a big data analytics perspective". In: *Big data research* 11, pp. 54–64.

Pearl, J., M. Glymour, and N.P. Jewell (2016). *Causal Inference in Statistics: A Primer*. Wiley. ISBN: 9781119186847.

Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf (2017). *Elements of causal inference: foundations and learning algorithms*. The MIT Press.

ProLeren (n.d.[a]). *E-learning ARI en rijwegen instellen*. URL: https://proleren.prorail.nl/mod/scorm/view.php?id=1118. (accessed: 31-01-2023).

– (n.d.[b]). *Goederen voor VM*. URL: https://proleren.prorail.nl/mod/scorm/view.php?id=8964. (accessed: 02-03-2023).

– (n.d.[c]). *Trein – gerelateerde afhandelingsstrategieën*. URL: https://prorailbv.sharepoint.com/teams/T2017_0030/Processen/TAD.aspx. (accessed: 31-01-2023).

ProRail (2018). *Preciezer plannen met PINT*. URL: https://prorailbv.sharepoint.com/sites/focusonline/SitePages/Preciezer-plannen-met-PINT.aspx. (accessed: 12-05-2023).

– (2021a). *Kerncijfers*. URL: https://www.jaarverslagprorail.nl/jaarverslag-2021/kerncijfers/. (accessed: 31-01-2023).

– (2021b). *Organisatie en activiteiten*. URL: https://www.jaarverslagprorail.nl/jaarverslag-2021/profiel/organisatie-en-activiteiten. (accessed: 31-01-2023).

– (2021c). *Verkeersleidingsposten*. URL: https://prorailbv.sharepoint.com/sites/vestigingen/SitePages/Verkeersleidingsposten.aspx. (accessed: 31-01-2023).

– (n.d.[a]). *Over Ons*. URL: https://www.prorail.nl/over-ons. (accessed: 31-01-2023).

– (n.d.[b]). *Rijwegen - basisinformatie: plan(regel)opbouw*. URL: https://prorailbv.sharepoint.com/sites/Achtergrondinfo-Treindienstleiders/SitePages/Rijwegen%20-%20basisinformatie%20-%20plan(regel)opbouw.aspx. (accessed: 31-01-2023).

Raghu, Vineet K et al. (2018). "Comparison of strategies for scalable causal discovery of latent variable models from mixed data". In: *International journal of data science and analytics* 6, pp. 33–45.

Spanninger, Thomas et al. (2022). "A review of train delay prediction approaches". In: *Journal of Rail Transport Planning & Management* 22, p. 100312.

Spirtes, Peter et al. (2000). *Causation, prediction, and search*. MIT press.

Triantafillou, Sofia and Ioannis Tsamardinos (2016). "Score-based vs Constraint-based Causal Learning in the Presence of Confounders." In: *Cfa@ uai*, pp. 59–67.

Tsamardinos, Ioannis, Laura E Brown, and Constantin F Aliferis (2006). "The max-min hill-climbing Bayesian network structure learning algorithm". In: *Machine learning* 65, pp. 31–78.

Ulak, Mehmet Baran, Anil Yazici, and Yun Zhang (2020). "Analyzing network-wide patterns of rail transit delays using Bayesian network learning". In: *Transportation Research Part C: Emerging Technologies* 119, p. 102749.

Wen, Chao, Zhongcan Li, et al. (2017). "Statistical investigation on train primary delay based on real records: evidence from Wuhan–Guangzhou HSR". In: *International Journal of Rail Transportation* 5.3, pp. 170–189.

Wen, Chao, Weiwei Mou, et al. (2020). "A predictive model of train delays on a railway line". In: *Journal of Forecasting* 39.3, pp. 470–488.

Wieringen, Richard van (Mar. 2019). "An Agent Based Approach for Train Traffic Control". MA thesis. University of Utrecht.

Wikipedia (2022). *Partial correlation*. URL: https://en.wikipedia.org/wiki/Partial_correlation.

Wu, Jianqing et al. (2021). "A Hybrid LSTM-CPS Approach for Long-Term Prediction of Train Delays in Multivariate Time Series". In: *Future Transportation* 1.3, pp. 765–776.

Zilko, Aurelius A, Dorota Kurowicka, and Rob MP Goverde (2016). "Modeling railway disruption lengths with Copula Bayesian Networks". In: *Transportation Research Part C: Emerging Technologies* 68, pp. 350–368.

Appendix A

Column Descriptions

This section elaborates on what the input data consists of and in which way the columns are modified. Firstly, we explain which columns were already present in the CSV file. Secondly, we explain which columns are created or kept after the preprocessing step described in section 6.2.

A.1 Dataframe from CSV file

The dataset is imported in the program as a CSV file. The columns that are kept, are denoted below, with an explanation of what the column means.

nvgb_verkeersdatum This column represents which date that the event corresponds to. The date is not changed when it is exactly 12 pm, but it depends on for which day the event was planned. If it represents a night train for the previous day, then the date is the previous day. If the train is an early morning train, then it will be denoted as the same day. The benefit of this, is that when a train crosses the 12 pm timestamp, it will still be grouped within the same day.

basic_treinnr_treinserie This denotes the serie number of the train. The definition of this is elaborated in Chapter 2.

basic_treinnr This denotes the train number of the train. The definition of this is elaborated in Chapter 2.

basic_spoor_simpel This denotes the platform number on which the train arrived at, and is filled if this event occurs at a station. Else, this column value is empty.

basic_drp *Dienstregelpunt* or timetabling point. The definition of this is elaborated in Chapter 2.

basic_drp_act Activity at the *dienstregelpunt* (Arrival, Departure, or Passage)

basic_plan This consists of the planned date and time that the train should perform the activity.

basic_uitvoer This is the real measured date and timestamp when the train performed the activity.

vklvos_plan_actueel This column is similar to the basic_plan column, but retrieved from a different source. If a train gets cancelled, this column will be empty, in contrast to the basic_plan. This value is therefore used to identify the cancelled trains.

prl_rijw_laatst_rijweg_wisselroute_maxsnelheid This column denotes the max speed that the train is allowed to drive. Under normal conditions this is filled with "baanvak", but if a speed change is required, this column value will be an integer consisting of the desired speed.
Note: The speed value column is only filled when a speed sign is passed at that drp. This means that for some rows, this column value is empty, because the train did not pass a speed sign at that point. Since the speed should be the same as the last registered speed sign, we use the forward-filling method to impute the missing values.

basic_treinnr_rijkarakter This denotes the type of train, namely Intercity, Sprinter, InterCity Express etc.

vklbammat_soorttype_uitgevoerd This denotes the type of trainset that was used for this train ride. When trains are combined, multiple trainsets are denoted in this column.

vklvos_bijzonderheid_drglsnelheid There is a planned speed at every section, which is called the baanvak speed. The baanvak speed differs per section and train type. This column is used to convert the values of the *prl_rijw_laatst_rijweg_wisselroute_maxsnelheid* that are marked as "baanvak" to the designated baanvak speed in integers.

infra_afgereden_secties This column denotes the section ID's that were used by the train.

donna_bd_kalerijtijd This denotes the strictly necessary travel time for a section. The planned travel time contains some slack.

stipt_oorzaak_treinnr This column denotes that if there was a train interaction on the railway, with which train number it was.

A.2 Dataframe after preprocessing

After preprocessing, some columns were removed and some columns were added. Finally, the Dataframe consists of the following columns. An arrow, $\rightarrow$, shows that a column is renamed. Some columns are kept the same, then an explanation is not included. The columns that are removed were used in the preprocess step, but not used afterward, hence they are not present anymore.

nvgb_verkeersdatum $\rightarrow$ date

basic_treinnr_treinserie

basic_treinnr

basic_spoor_simpel

basic_drp

basic_drp_act

basic_plan

global_plan This is the basic_plan value, rounded up by minutes.

prl_rijw_laatst_rijweg_wisselroute_maxsnelheid $\rightarrow$ speed This column is modified by first using the forward filling method to impute missing values. Then the values with "baanvak" are converted to the baanvak value in integers, using the corresponding *vklvos_bijzonderheid_drglsnelheid* column value.

basic_treinnr_rijkarakter

vklbammat_soorttype_uitgevoerd

infra_afgereden_secties $\rightarrow$ sections

delay This denotes the delay of the train and is calculated by: $delay = basic\_uitvoer - basic\_plan$. Note: delays can thus be negative as well.

buffer This denotes the planned buffer time that a train has, when waiting at a station. This is calculated by: $delay_i = basic\_plan_i - basic\_plan_{i-1}$ and only if the activities have the same train number and are at the same station. Else, buffer = 0

travel_time This denotes the travel time the train has between two activities. This is calculated by: $travel\_time_i = basic\_plan_i - basic\_plan_{i-1}$ and only if the activities have the same train number. Else, travel_time = 0

slack Slack can be retrieved by $(planned)travel\_time - donna\_bd\_kalerijtijd$.

stipt_oorzaak_treinnr

Appendix B

Description of the search functions in the Causal-learn library

The implementation of the structure learning part is conducted by using the causal-learn library, version 1.3.1. The causal learn library consists of several discovery algorithms. The constraint-based causal discovery methods accept a parameter called *background_knowledge* to use prior knowledge for the discovery algorithm. The *background_knowledge* parameter consists of a *BackgroundKnowledge* object, which stores the *required_rules_specs* and the *forbidden_rules_specs*. These rules contain of a set of tuples of nodes, for which the rules apply to. The forbidden rules can be added by the function `add_forbidden_by_node(node1, node2)` and the required rules can be added by the function `add_required_by_node(node1, node2)`. To check if the edge between two nodes is forbidden, the function `is_forbidden(node1, node2)` is used. To check if the edge between two nodes are required, the function `is_required(node1, node2)` is used [Causal-learn 2022a].

B.1 PC-algorithm

The PC-algorithm with background knowledge is implemented in the library as follows. First, the PC algorithm performs a skeleton discovery algorithm. The following actions are performed individually per node. If an edge is marked as forbidden in both ways, the boolean *knowledge_ban_edge* set to true. As a consequence, the edge is removed from the graph in both directions. Then, the independence tests are performed on the node

and its remaining neighbors and if the result is larger than the alpha value, this edge will be removed [Causal-learn 2022c]. The consequence of this skeleton discovery function is that the required edges are not taken into account, and required edges may be removed because of a significant independence test.

The PC algorithm then proceeds with orienting the edges, first by using the background knowledge. If node1 and node2 is forbidden, then orient the edge as node2 $\rightarrow$ node1. If node1 and node2 is required, then orient the edge as node1 $\rightarrow$ node2. Then the edges are oriented by looking for v-structures, as described in step 3 of the PC algorithm. Lastly, the method of orienting the remaining edges as described by Meek (2013) is performed to orient some other edges.

B.2 FCI-algorithm

The FCI-algorithm first performs a skeleton search by means of a Fast Adjacency Search (FAS). FAS consist of a searchAtDepth0 function and a searchAtDepth for the other depths.

Depth0:
The algorithm does the following between every two nodes. It first performs the independence test. If the independence test is larger than the alpha value and the edges are not required, the separating set between the two nodes is the empty set. If the variables don't contain a forbidden edge between them (in both directions), the nodes include each other in their adjacency list. This step creates a first skeleton of the graph, by excluding all forbidden edges in the adjacency lists.

Depth-x:
For each node a and one of their adjacent nodes b, all possible combinations of the adjacent edges of a (excluding b) of length x are created, which functions as the conditioning set. It again performs the independence test, conditioning on the conditioning set. If the independence test is larger than the alpha value and the edge between a and b is not required, the separating set between the two nodes will be updated with the conditioning set and the adjacency list will be updated by removing the nodes both ways out of their own list.

The FAS function returns a tuple of the found General Graph and the corresponding separating set. The FCI algorithm then proceeds with reorienting the edges, by first orienting all edges as o-o, and then orienting according to the background knowledge. This is the same method as described in the PC algorithm. After this step, the edges are further oriented by looking for v-structures. Then, the algorithm proceeds with step 4 of the FCI algorithm, by looking for possible separating sets. Every edge that may be removed is checked if it is not required by the background knowledge before adding to the possible separating set. Then the edges in this set are removed and the separating set is updated accordingly. Lastly, steps 5 and 6 of the FCI algorithm are performed, where the background knowledge determines the orientation of some edges as described previously and other edges are oriented by identifying v-structures and remaining orienting techniques [Causal-learn 2022b].

Appendix C

Create BackgroundKnowledge

In this section, the description is given on how to translate the TrainRideObjects to a BackgroundKnowledge class. As described in Section 7.2.1, there are 3 types of edges: forbidden, required and possible edges.

In our algorithm, all edges are first marked as forbidden. The reason to do this is that most of the edges are forbidden and, this way, not all nodes need to be compared with each other to determine if they are actually forbidden. This speeds up the computation time. When an edge is marked later as "required" or "possible", this "forbidden" label will be removed.

The next step is to find the required edges. The schedule of TrainRideObjects is ordered per train number and timewise. The only check that needs to be performed is to check if the train number at index $i + 1$ is the same as at index i. If this is the case, we mark the edge as required. Resulting, all subsequent events with the same train number are marked required. Events that are not adjacent are not required.

The last step is to find the possible edges. Only trains that are at the same *drp* within 15 minutes are marked in our model as "possible", by removing the "forbidden" label. The first step is to create a dictionary with each station as key and as value the tuple, (timestamp, TrainRideObject). An example is: $\{Bkl : [(8:15, TrainRideObject1), (13:05, TrainRideObject2)]\}$. Only within the list per station, the trains are evaluated to each other, making our search space smaller. The list of trains per station is ordered timewise. The trains are evaluated by comparing their planned time. If that lies between 15 minutes, the edge between these events is marked as "possible", by removing the edge as "forbidden". The global algorithm is shown in Algorithm 1

Algorithm 1 Find possible edges

```
1: station_dict # our station dictionary
2: bk # our current background knowledge
3: for station, train_list in station_dict do
4:     train_list = sort_on_timestamp(train_list)
5:     for train_index in range(len(train_list)) do
6:         (train_time, train) = train_list[train_index]
7:         for other_train_index in range(train_index +1, len(train_list)) do
8:             (other_train_time, other_train) = train_list[other_train_index]
9:             if (other_train_time - train_time) <= 15 · 60 # seconds then
10:                bk.removeForbiddenDependency (train, other_train)
```

Appendix D

Improve computation speed for BackgroundKnowledge class

When using the causal library, the computation speed increased significantly when adding a BackgroundKnowledge class that consists of many forbidden or required edges. These edges are stored in the *required_rules_specs* and *forbidden_rules_specs* variables, which consist of a set of tuples. An example is shown below.

$$dict\_forbidden = (("train1", "train2"), ("train1", "train3), ("train3, "train1"))$$

The functions `is_forbidden()` and `is_required()` performs an exhaustive search that loops through all the *required_rules_specs* or the *forbidden_rules_specs* variable. As a consequence, the more background information is added, the longer this search takes.

This slower BackgroundKnowledge class is in this research replaced by a new class: FasterBackgroundKnowledge, which inherits from the BackgroundKnowledge class. It consists of two dictionaries, one for the forbidden edges and one for the required edges. The dictionary key is the name of one train, and the corresponding value is a list of the names of the trains that should have a forbidden edge with that train. The previous example of the *dict_forbidden* variable is translated into the corresponding dictionary as shown below

$$dict\_forbidden = \{"train1" : ["train2", "train3"], "train3" : ["train1"]\}$$

When checking if node1 should have a forbidden edge with node2, the function *is_forbidden(node1, node2)* checks if node1 exists in the dict and retrieves the list of forbidden edges for node1 and checks if node2 is present. Using this function rather than the old *is_forbidden(node1,*

node2) function, a smaller list has to be searched, which makes this version of the BackgroundKnowledge class faster.

Note: in the file `fas.py`, line 286 of the library, there is a check if the provided BackgroundKnowledge is of the correct type, by stating `type(knowledge) == BackgroundKnowledge`. This should be changed to `isinstance(knowledge, BackgroundKnowledge)`; the type checking will still be executed, and due to inheritance, no errors will occur.

Appendix E

Improvement of the FAS-function

As described in Appendix B, the FAS-function consists of two main functions, `searchAtDepth0()` and `searchAtDepth()`. After the improvement described in Appendix D, the FAS() function was still slow. To put it in perspective: for a dataset that consists of 2427 variables and in 70 rows per variable, it took around 44 hours to complete. To speed up the FAS-function, the weaknesses of this function are analyzed and listed below.

searchAtDepth0()

This function is slow, since it iterates over all node combinations and performs independence tests between them. After this, it checks if those edges were required and if they were not forbidden. Only if they were not forbidden, the nodes are added in the adjacency list. If these nodes were forbidden, the adjacency list is not updated.

Solution: only keep the possible and required edges in the graph, by creating the adjacency list only based on not forbidden edges. Then perform independence tests between edges in this graph, since there is now a reduced search space and the redundant independent tests are not performed anymore.

searchAtDepth()

To perform an independence test between two nodes, all possible sets to condition on are determined, the significance of independence is calculated and then checked if this edge is not required. If this edge is required, then nothing happens. This makes the previous work redundant. It would be better to first check if the edge is not required and then perform the rest of the code.

The adjacency list is only filled with nodes that are not forbidden. However, when looking for possible parents, each node combination is checked whether they are not forbidden. Looking at the sequence of the code and the structure of the Background-Knowledge that is created for this research, the edges are never forbidden, so this check is redundant.

With these remarks, a new FAS() function is created. This function also consists of two parts. The first part creates a graph with all edges connected that are not forbidden. The second part performs the independence tests between the present adjacencies of the nodes per depth. Here it first checks if the adjacent nodes are required and if so, it does not check for independence and continues with the next adjacency. In this phase, there is no check for forbidden edges necessary, since the forbidden edges are already pruned in the first phase. Comparing the results of the old and new FAS function, the resulting graphs were the same. The computation time decreased from around 44 hours to around 2 minutes.

Algorithm 2 FCI

```
Input: dataset matrix, background
Output: SCM graph with directed edges
Variables:
adjacencies: Dict,
separating set : Dict,
nodes : List

For each column, create a node and add to node_list
# Create a skeleton graph with all edges that are not forbidden:
for i ← 0 to N do
    for j ← i + 1 to N do
        if isforbidden(nodes[i],nodes[j]) and isforbidden(nodes[j],nodes[i]) then
            Continue
        else
         mark nodes[i], nodes[j] as adjacency
         mark nodes[j], nodes[i] as adjacency
maxDepth = 20
for d ← 0 to maxDepth do
    enough_adjacencies_for_depth = False
    for i ← 0 to N do
        adjx = adjacencies of node i
        for j ← 0 to len(adjx) do
            if isRequired(i,j) then Continue
            Find all possible subsets from adjx/nodes[j] as condition set.
            for all possible_subsets do
                enough_adjacencies_for_depth = True
                do independence test
                if p_value > threshold then
                    remove adjacency nodes[i], nodes[j]
                    remove adjacency nodes[j], nodes[i]
                    add separating set for nodes[i], nodes[j] with conditioning set
    if not enough_adjacencies_for_depth then
        Break
```

Appendix F

MV-Fisher-z independence test

The used independence test method is the Missing Value Fisher-z independence test, used from the Causal-learn library. This is the same as the general Fisher-Z independence test, but rows that consist of empty values are removed. If the Dataframe will then be empty, which means that there is no overlap in trains for any day, we return 1 by default, since they are by definition independent of each other. The Fisher-Z test uses the Fisher partial correlation method and is implemented by the causal-learn library as follows:
1) It first calculates the Pearson correlation coefficient matrix R, for which each matrix-value is calculated as follows.

$$R_{i,j} = \frac{C_{i,j}}{\sqrt{C_{i,i}C_{j,j}}}$$

2) Then it computes the inverse of the matrix, using the function *np.linalg.inv*.
3) The correlation between the Xth value and the Yth value is calculated again by using the Pearson correlation coefficient on the inverse matrix, which results into the partial correlation value $\hat{\rho}_{XY \cdot Z}$, which includes the conditioning set z in the value.

$$\hat{\rho}_{XY \cdot Z} = -\frac{C^{-1}_{i,j}}{\sqrt{C^{-1}_{i,i} \cdot C^{-1}_{j,j}}}$$

4) Then this ρ-value is converted to a Z-value by the following Fisher Transformation Formula [Glen n.d.]:

$$Z = \frac{1}{2} \ln(\frac{1 + \hat{\rho}_{XY \cdot Z}}{1 - \hat{\rho}_{XY \cdot Z}})$$

To convert the Z-value to a p-value, several steps are preformed. First, the Z-value is converted [CLeaR 2022]:

$$X = \sqrt{(N - |z| - 3)} \cdot |Z|$$

According to Wikipedia (2022), value X can be tested against a null hypothesis that the partial correlation is 0. The hypothesis can be rejected if

$$X > \Phi^{-1}(1 - \frac{\alpha}{2})$$

Φ is the cumulative distribution function of a Gaussian distribution. If this formula returns the boolean *true*, the tested values are independent of each other. The formula can be rewritten by the following steps:

$$\Phi(X) > 1 - \frac{\alpha}{2}$$

$$-1 + \Phi(X) > -\frac{\alpha}{2}$$

$$1 - \Phi(X) > \frac{\alpha}{2}$$

$$2 \cdot (1 - \Phi(X)) > \alpha$$

This last formula is included in the library, namely, stating that p-value $= 2 \cdot (1 - norm.cdf(abs(X))$ and $p\_value > \alpha$ must be true to be independent.

Appendix G

Graph to a Neural Network input

This section describes the method how the causal graph is translated to an input matrix for the Neural Networks. The transformation from graph to input matrix is done in 3 steps. Each step is explained in one section.

G.1 Convert the graph to a NodeAndParents class

The first step is to convert the graph to a class that describes the parents per node, called the NodeAndParents class. This class consists of several variables that are each a TrainRideObject. It captures which TRO was the previous event, which was the 2nd previous event and which train was at the same platform as the current train. The other non-specified parents are captured in a list of dependencies, ordered by their planned time.

The algorithm goes as follows: per node in the graph, its parents are retrieved. For each of the parents, it is checked which type of parent it is (previous event, dep0_platform or another non-specified parent). If it is the previous event, then its parents are also called, such that the 2nd previous event is also found. Then, the previous event and the second previous event are included in the NodeAndParents class. If the parent is a dep0_platform, found by comparing both of the TRO their station and platform, then this dep0_platform variable is set to the found TRO. If the parent is not either of those two cases, it is included in the list of non-specified parents. After every parent is assigned to their type, the list of non-specified parents is ordered by their planned time. The result of this algorithm is a list of NodeAndParents classes. In total, there are l NodeAndParents class in the list, where l is the amount of nodes in the graph. An example of this algorithm is shown in Figures G.1 and G.2.

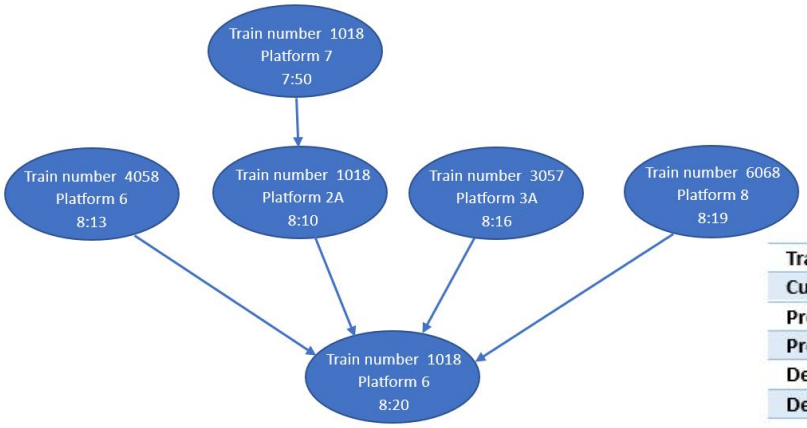

FIGURE G.1: Input graph

Train type	TrainRideNode
Current train	Train 1018 at platform 6
Prev_event	Train 1018 at platform 2A
Prev_prev_event	Train 1018 at platform 7
Dep0_platform	Train 4058 at platform 6
Deps	Trains [3057, 6068]

FIGURE G.2: Result of algorithm

G.2 Extract delays from the NodeAndParents class

The next step uses the NodeAndParents class and the created TrainRideObject matrix introduced in Section 6.3 to find all the delays of each day and include the delays of the parents as well. The result of this step is a list with a class denoting the input features for the Neural Network, called NN_input_row class.

The working of how the delays are included per NodeAndParents object as input is illustrated in Figure G.3. The first matrix denotes the matrix of the delays of the TrainRideObjects. The arrows denote which columns need to be included, as they are the parents of a specific train. The correct columns are found by looking at the NodeAndParents object, which includes the TRO of all parents. Each TRO contains a variable at which index it is present in the TRO matrix, hence the colored arrows.

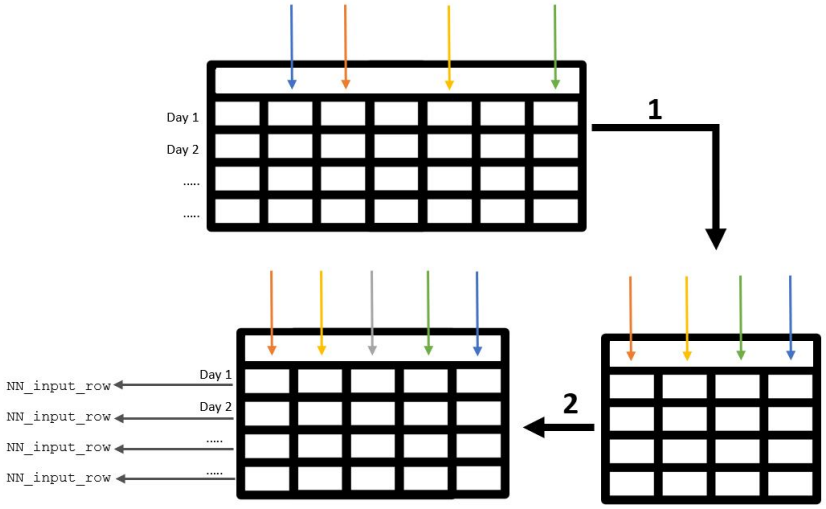

FIGURE G.3: Capturing and positioning of the correct columns

The first step is to only keep the desired columns, resulting in the second table. The second step is to impute columns that are denoting a parent that is not present for that specific train (in the figure denoted as the gray arrow). For example, there is not an interacting train that is at the same platform as our train, this column cannot be extracted from the TRO matrix. Such column is added in the second step at the correct position. The final result is a matrix with the columns that are always at the same index. The order is: current train, previous event, 2nd previous event, dep0-platform, dep1, dep2, and dep3. Each row is a parameter for a NN_input_row class. In total, there are $k \cdot l$ NN_input_row classes, where k is the amount of days and l is the amount of nodes in the graph.

G.3 NN_input_row class to Dataframe

In this section, the function NN_input_class_to_df() is explained, where the list of NN_input_rows are transformed to a Dataframe that can be used as input for the Neural Networks. The function loops through the list of the NN_input_rows and preprocesses the data such that it becomes a Dataframe row. Some columns are changed to a one-hot encoding, and thus, some columns are expanded. For example, the day_of_the_week column is changed to 5 columns: day_of_the_week0 to day_of_the_week4. Each row is appended to the Dataframe, resulting into a Dataframe with $k \cdot l$ rows, where k is the amount of days and l is the amount of nodes in the graph.

Appendix H

Derivation of the loss function for a Laplace distribution

The PDF of the Laplace distribution is the following [1] [2], with variance being $2b^2$:

$$f(x|\mu, b) = \frac{1}{2b} exp(-\frac{x-\mu}{b})$$

By applying the conditional log likelihood ($\sum_{i=0}^{m} \log p(y^{(i)}|x^{(i)}; \theta)$), it can be rewritten as

$$\sum_{i=1}^{m} -\log 2b - \frac{x^{(i)} - \mu}{b}$$

Which can be rewritten as:

$$-m \log 2b - \sum_{i=1}^{m} \frac{x^{(i)} - \mu}{b}$$

With μ being $\frac{\sum_{i=1}^{m} \hat{x}}{m}$, we can rewrite the formula to:

$$-m \log 2b - \sum_{i=1}^{m} \frac{||\hat{y}^{(i)} - y^{(i)}||}{b}$$

For this case then maximizing the log likelihood yields the same estimate as minimizing the MAE ($= \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||$) According to this derivation, the MAE would be the most suitable loss function for the Laplace distribution.

[1] https://math.stackexchange.com/questions/922521/deriving-mean-and-variance-of-laplace-distribution

[2] https://en.wikipedia.org/wiki/Laplace_distribution

Appendix I

Results Amsterdam-Utrecht railway

I.1 Number of Neural Networks and amount of train-, test-, and validation data

The training set consists of 550,230 samples: 440,184 samples for testing and 110,046 samples for validation. The test set consists of 137,558 samples.

The set of fine-tuned NNs are bucketed per (*drp*, train series) and consists of 477 Neural Networks. The maximum number of samples to train the NN per bucket is 4390 and the minimum is 17. The mean is 1153 and the median is 651 samples. The amount of samples to test on per fine-tuned NN is maximal 1124, minimal 1, on average 288, and the median amount of samples is 165.

I.2 Model evaluation on the total test set and per station

The fine-tuned Neural Networks are tested and evaluated with two metrics: the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). The MAE on the total dataset is 10.03 and the RMSE is 36.87. The errors per *drp* in terms of MAE and RMSE are shown in Figure I.1 and I.2. The largest errors at Asd (Amsterdam Central Station) and Ut (Utrecht Central Station) can be explained by the fact that both stations are large stations consisting of a large yard, allowing for more train interaction. The spike of the RMSE value at Asdma (Amsterdam Muiderpoort aansluiting) can be explained by the fact that there are 3 large estimation errors (2403.0466, 2463.8425,

and 3140.458 seconds). Removing these outliers, the estimation errors MAE and RMSE at Asdma would be 6.05 and 22.09 respectively. Also, removing a large outlier for Ashd (Amsterdam Holendrecht) would change the MAE and RMSE to 6.12 and 12.56 respectively.

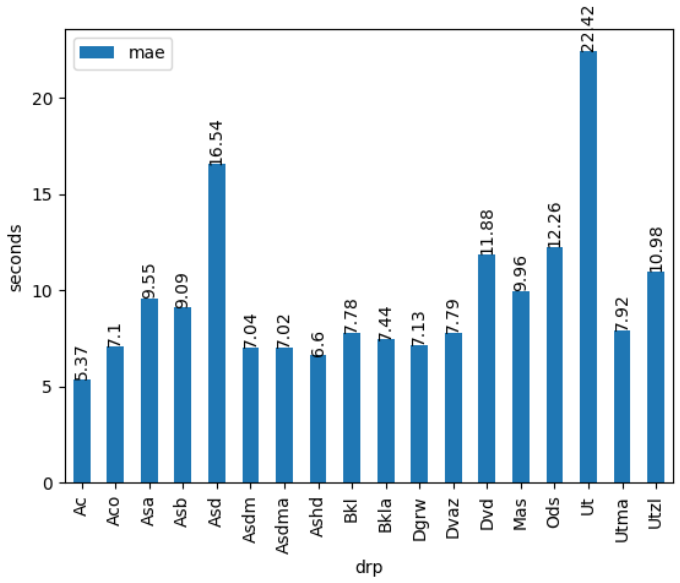

FIGURE I.1: MAE per drp

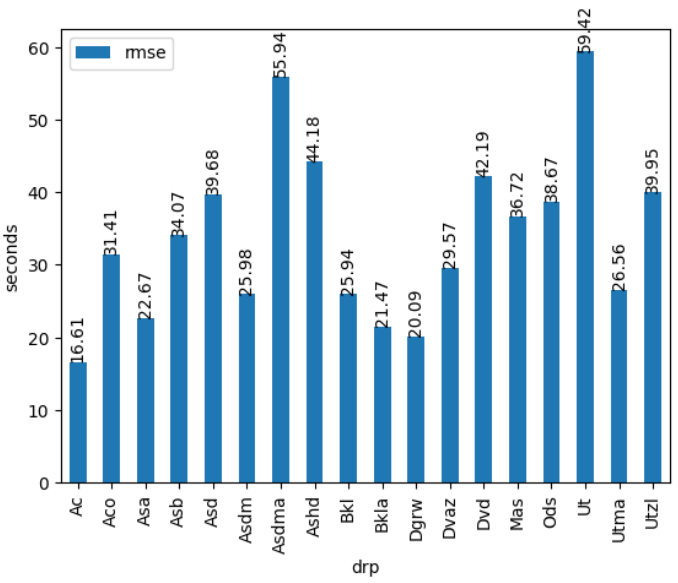

FIGURE I.2: RMSE per drp

I.3 Testing NNs for Amsterdam-Utrecht with more specific test sets

In this section, the results are further analyzed by focussing on the events for which it is known that a train interaction took place. When ProRail registered a train interaction, the column "cause" is filled with the corresponding train number. A downside to this column is that if a train only hindered another train for even one second, this train will be included in the column. Such interactions are too small to be noticed by our algorithm. The dataset is therefore further reduced by removing rows where our algorithm did not observe a possible interaction of non-trivial parents (i.e.: dep0 to dep3 are empty). This subset is called: subset of rows including non-trivial parents and where ProRail found an interaction.

In order to test the effect of the non-trivial parents included in our model, our model is tested against another model. Our model is trained and tested with data containing the columns about non-trivial parents. The other model is trained and tested on the same sets, but the columns about the non-trivial parents are not present. In this chapter, we will refer to the model with the columns that includes the non-trivial parents as `*our*

model' and the other model without the columns regarding non-trivial parents as the `*baseline model*'. The results are denoted in Table I.1.

Test-set	MAE	RMSE	# of test-samples
Our model: Total test set	10.03	33.75	137,558
Baseline model: Total test set	9.99	37.47	137,558
Our model: where ProRail found an interaction with the columns non-trivial parents included	53.11	86.65	446
Baseline model: where ProRail found an interaction excluding the columns non-trivial parents	59.93	96.45	446

TABLE I.1: Results of the Asd-Ut line comparing it to another trained model

Appendix J

Results Utrecht-Eindhoven railway

J.1 Number of Neural Networks and amount of train-, test-, and validation data

The training set consists of 502,233 samples: 401,786 samples for testing and 100,446 samples for validation. The test set consists of 125,558 samples.

The set of fine-tuned NNs are bucketed per (*drp*, train series) and consists of 309 Neural Networks. The maximum number of samples to train the NN per bucket is 4114 and the minimum is 16 The mean is 1520 and the median is 1607 samples. The amount of samples to test on per fine-tuned NN is maximal 1111, minimal 0, on average 406, and the median amount of samples is 430.

J.2 Model evaluation on the total test set and per station

The fine-tuned model is tested and evaluated with two metrics: the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). The MAE on the total dataset is 19.94 and the RMSE is 50.85. The errors per *drp* in terms of MAE and RMSE are shown in Figures J.1 and J.2. The largest errors at Ut (Utrecht Central Station) and Ehv (Eindhoven Central Station) can be explained by the fact that both stations are large stations consisting of a large yard, allowing for more train interaction. Two prominent outliers in the predictions contributed to the elevated RMSE value of Utvr (Utrecht Vaartsche Rijn). Removing these two predictions, the MAE and RMSE of Utvr would be 7 and 25.11 respectively.

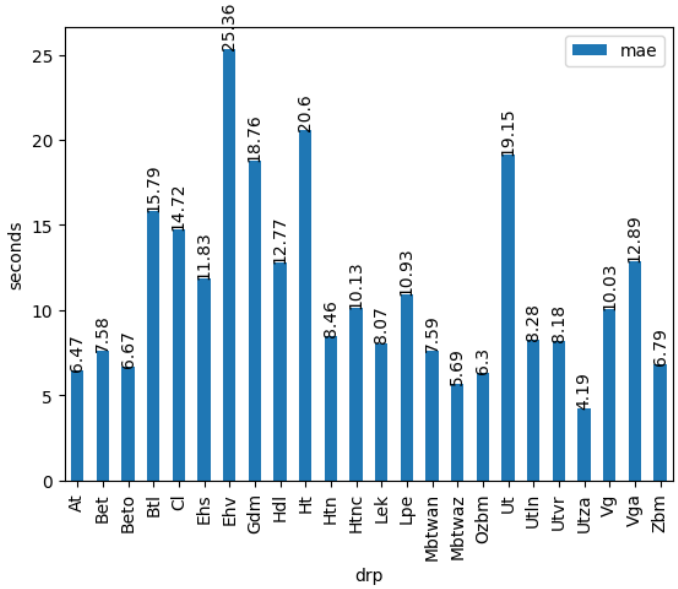

FIGURE J.1: MAE per drp

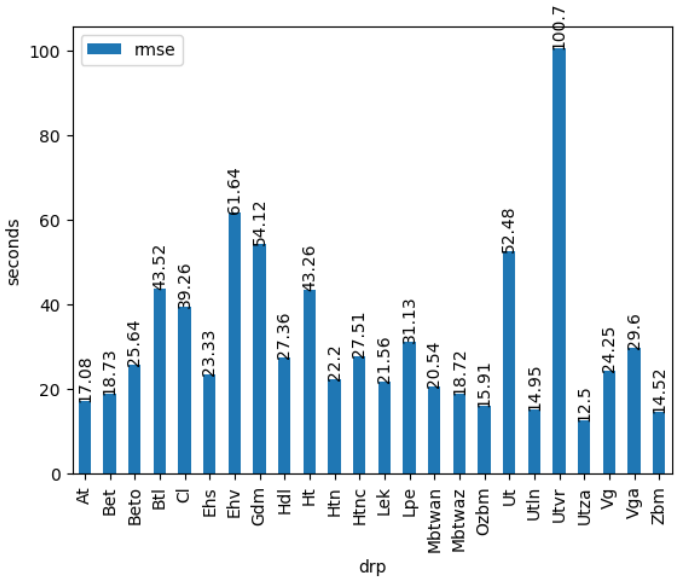

FIGURE J.2: RMSE per drp

J.3 Testing NNs for Utrecht-Eindhoven with more specific test sets

In this section, the results are further analyzed by focussing on the events for which it is known that a train interaction took place. When ProRail registered a train interaction, the column "cause" is filled with the corresponding train number. A downside to this column is that if a train only hindered another train for even one second, this train will be included in the column. Such interactions are too small to be noticed by our algorithm. The dataset is therefore further reduced by removing rows where our algorithm did not observe a possible interaction of non-trivial parents (i.e.: dep0 to dep3 are empty). This subset is called: subset of rows including non-trivial parents and where ProRail found an interaction.

In order to test the effect of the non-trivial parents included in our model, our model is tested against another model. Our model is trained and tested with data containing the columns about non-trivial parents. The other model is trained and tested on the same sets, but the columns about the non-trivial parents are not present. In this chapter, we will refer to the model with the columns that includes the non-trivial parents as `*our model*' and the other model without the columns regarding non-trivial parents as the `*baseline model*'. The results are denoted in Table J.1.

Test-set	MAE	RMSE	# of test-samples
Our model: Total test set	11.78	43.93	125,558
Baseline model: Total test set	20.09	53.53	125,558
Our model: where ProRail found an interaction with the columns non-trivial parents included	41.21	82.14	752
Baseline model: where ProRail found an interaction excluding the columns non-trivial parents	59.65	100.29	752

TABLE J.1: Results of the Ut-Ehv line comparing it to another trained model