Railway delay management with offline passenger rerouting

Minimizing the total passenger delay

M.S. van Adrichem

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Railway delay management with offline passenger rerouting

Minimizing the total passenger delay

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

M.S. van Adrichem

April 13, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Delay Management (DM) is a proven to reduce the average passenger delay. However, the current method of DM either uses fast basic or slow accurate passenger rerouting. The former has a downfall in that it is inaccurate since it assumes that passengers delay is the periodicity of the train of the original transfer, while in reality, faster alternatives are often available. The latter reroutes passengers simultaneously with the delay management resulting in computation times larger than the time window of a Train Dispatcher (TD) to make a decision. This thesis proposes an alternative method of DM which calculates the alternative routes before the Delay Management optimization using a modified Dijkstra algorithm. This method outperforms the fast basic approach to reducing passenger delay with a similar computation time, thereby outperforming the slow accurate approach.

Table of Contents

"You may delay, but time will not."

— *Benjamin Franklin*

Chapter 1

Introduction

Railway transport is an effective means of transportation with a lot of upsides for society. Passengers who experience train delays might avoid this form of transportation in the future. Therefore, this thesis looks into minimizing the effect of delayed trains on the passengers, thereby improving the passenger experience and reducing the avoidance of public transportation.

This introduction starts with the relevance of the thesis in Section 1-1, followed by the current Dutch delay management approach in Section 1-2. Thereafter, Section 1-3 is on how Delay Management (DM) implemented as a Decision Support System (DSS) can support Train Dispatchers (TDs) in reducing the effect of delays. The fourth section discusses the literature gap. Section 1-5 is on the problem description which includes the research questions followed by a section on the scope of this thesis. At last, Section 1-6 gives the outline and structure of the thesis.

1-1 Relevance of the research

People travelling by railway instead of by car has a lot of advantages for society. The major advantage is travelling by train produces fewer emissions than travelling by car and therefore better for the human health and the environment. A reduction in car use also has other advantages:

1. Fewer traffic jams; improved traffic flow and thereby also reducing emissions.
2. A reduction in the probability of a traffic incident.
3. Fewer parking problems.

Therefore it is generally more beneficial for society if people use public transport more often. However, in reality, a big part of the population still prefers travelling by car.

The choice of mode of transportation for travelling long distances is made by opting for the mode with the best relation between the three budgets: time, money and effort. The specific choice between travel by car or train is based on the difference in reliability, travel time, ease, comfort, experience and cost, with travel time being the most important factor in this decision [30].

A large group of people do not even consider the train a viable option. This is a result of the perception of car travellers; on average their perception of public transport travel time far exceeds the objective values. There are several explanations for the distorted perception [29];

1. The people have a lack of knowledge of the public transport.
2. Conscious or unconscious bias for justifying the choice for car use.
3. Bias as a result of past negative experience.

The focus of this thesis is on the last point, specifically the negative experience as a result of delays. People have become more sensitive to time and live higher paced lives. This results in delays having the biggest impact on their experience [30]. Train delays can not be prevented completely so reducing the negative experience as a result of delays is the only option. Reducing the negative experience will result in improved time perception and an increase in the likelihood of people using the train.

There are various approaches for reducing the negative experience of train delays. One approach is to provide the passengers with clear information to reduce the uncertainty and impact of the delay. In the Netherlands, this is already accomplished by accurate information on the website or in the app of the Dutch Railways (Dutch: Nederlandse Spoorwegen) (NS) or 9292[1].

Another approach is to influence the experience of delayed passengers with the environment, by providing sheltered waiting and seating areas and food and refreshment facilities or influencing passengers with colour, light, music, advertising and infotainment [30]. This approach softens the waiting experience but there is another approach that has a more direct impact on the negative effect of delays.

The most direct method for decreasing the negative experience is to manage a network with a delayed train such that the overall impact on passengers is minimal. The following section describes the current dutch management approach, followed by a section on how DM as a tool can be a useful addition to the current management approach.

[1] 9292 is a platform used for route planning in the Netherlands, has up to date information.

1-2 Current Dutch delay management approach

The current (Dutch) approach to managing the effect of a delayed train is to use the Train Service Handling Document (Dutch: Treindienst Afhandelings Document) (TAD). The TADs are unique and from the perspective of one train, containing the dispatching rules for the most common situations [25]:

1. How long a train can wait for a connecting train.
2. Where to short-turn[2] a train.
3. Provide resolution rules if there is an ordering conflict between trains. The TAD states the order of the train for different delay scenarios.

The TADs are written by ProRail[3] in collaboration with the Dutch Railways (Dutch: Nederlandse Spoorwegen) (NS) and other railway operators. There are some downsides to this TAD method [25] [1]:

1. Not every conflict is covered in the TAD, solving the conflicts not stated within the TAD heavily relying on the skill and experience of the Train Dispatcher (TD).
2. The TAD are unique for each train. At each stop, different delay scenarios are listed. Each scenario states how to interact with three different trains. Large or multiple delays can result in more trains requiring rescheduling, how the other trains should reschedule is not included in the TAD. In these cases, the TD needs to rely on experience to reschedule the other trains.
3. The TAD prioritizes trains running on time, while some trains might be worth prioritizing on the bases of the group size of passengers for which it is significant.
4. The TADs have to be reevaluated and rewritten each time a new timetable is released. Since this is done in collaboration with other parties this results in extra work.
5. The TAD currently in use are the result of multiple iterations based on calculations but also experience. Implementing large changes in the current timetable is therefore hard since it requires a lot of new or rewritten TADs.

The following section explains how Delay Management (DM) in the form of a Decision Support System (DSS) can be used besides the Train Service Handling Document (Dutch: Treindienst Afhandelings Document) (TAD) to improve the passenger experience.

[2] in case of a disruption the train can not complete the scheduled line and turns around before skipping some stations

[3] ProRail is responsible for the maintenance and traffic control of the railway network.

1-3 Delay management as support tool

The previous section was on the current management approach of delay with delays; Train Service Handling Document (Dutch: Treindienst Afhandelings Document) (TAD). This section focuses on how to improve the current management situation by providing a Delay Management (DM) tool which can be implemented next to the current TADs to compensate for its shortcomings. The section starts with the basics of DM followed by how it can be implemented using a Decision Support System (DSS).

1-3-1 Basics of delay management

Minimizing the effect of delayed trains on passengers journeys is in the literature referred to as DM. This approach focuses on transfers, passenger flow data and minimizing the average passenger delay while maintaining a feasible network. The minimization of the average passenger delay takes into account passengers with transfers and their delays in case they miss their transfer. On the other hand, the main focus of TD is deviating as little as possible from the timetable. TDs relay on set rules for maintaining transfers, prioritizing the robustness of the network. With DM tools, the feasibility of the network can be simulated. The transfer is maintained with larger delays in comparison to the traditional rules.

Delay Management (DM) is first introduced by Schöbel (2001) [26]. It uses Mixed Integer Linear Programming (MILP)[4] to reduce the total passenger delay while maintaining a feasible network. Schöbel also introduces the so-called wait-depart decision; should a connecting train either **wait** for a delayed feeder train and propagate the delay in the network or **depart** on time causing the transferring passengers to miss their connection. Waiting results in a delay for passengers on the connecting train, also propagating the delay further in the network. This could result in other wait-depart decisions or an infeasible network. DM is further in-depth explained in Section 1-3.

1-3-2 Decision Support System

DM applied in the form of a DSS will reduce the passengers average passenger delay [5]. Figure 1-1 shows a schematic representation of the DSS. Advantages of a DSS are that [28]:

- It can be applied to many management problems.
- It reduces the required level of experience and skill of the TDs.
- Decisions can be simulated before they are executed, thereby reducing the need for risk-averse decisions.[5]
- It is a good support tool in case the network is highly disturbed, the system can handle and display a lot of information.

A requirement for the DSS is a fast computation time such that the TD can run various scenario's before the deadline of implementation; decisions are made under time pressure.

[4] see Section 2-2-2 for explanation of MILP

[5] Unnecessarily breaking a connection to ensure the feasibility of the network.

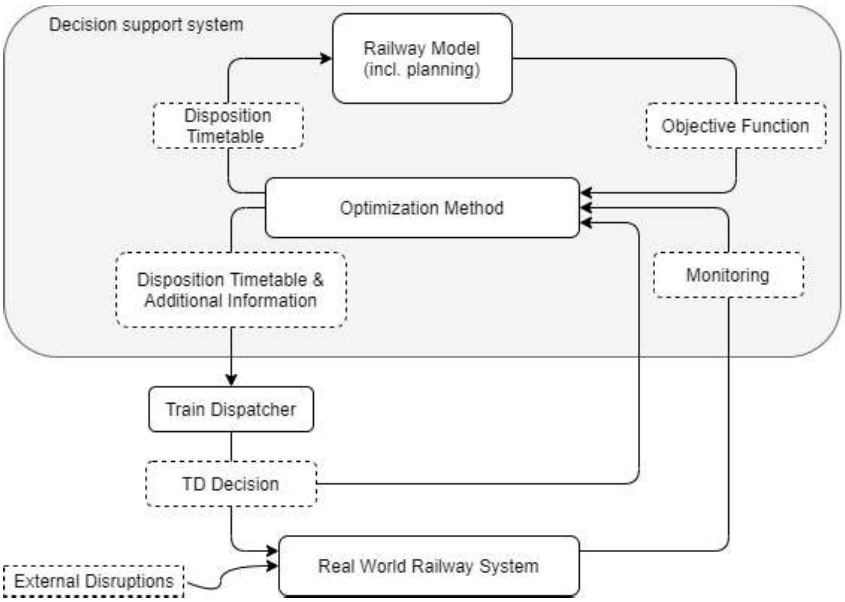


Figure 1-1: Decision support system

1-4 Research gap

This section is on the research gap this thesis will try to fill. This section starts with a reasoning for the lack of research in this field followed by the gap between the complex and fast models in DM.

Reasoning for the lack of research

Since passenger satisfaction plays such an important role in the choice of mode of transportation one would assume DM would be a well-researched field. However, the majority of literature focuses not on minimizing the passenger delay but on minimizing the train delay, also referred to as Real-Time Rescheduling (RTR). RTR major focus is on feasibility with a microscopic model approach [19][6]. The lack of literature on DM results in the potential for a lot of innovation.

The lack of research in DM is possibly the result of passenger flow information being scarcely available. When prioritizing between different trains TDs work with estimations based on their experience and passenger countings in the past [4]. With the introduction of the OV-chipcard[7] passenger flow data will become available.

Accurate vs fast Delay Management models

Where the majority of the approach in DM is the same, differentiation occurs in the formulation of the objective function and the addition of constraints. The focus of this thesis is on

[6] this is a generalization some literature uses a combination of DM, disruption management or RTR
[7] A contactless smart-card used for payments in the public transport in the Netherlands

the gap in the literature regarding accurate passenger routing. Literature in DM either uses a basic version of passenger rerouting or accurate passenger rerouting.

Basic: The majority of literature in DM applies basic passenger routing. Passengers whose transfer is broken take as an alternative route the next train of the train line of the connecting train. This simplification does not take into account the possibility of an alternative shortest path with an earlier arrival time, maintaining transfers longer than necessary. The basic version is further explained in Section 2-3-1.

Accurate: During research only two papers on active passenger rerouting where found [12] [20]. These papers introduce a lot of new variables and constraints such that alternative passenger routes are computed simultaneously to the optimization of the average passenger delay. This increase results in a large increase in computation time. The complex online rerouting is further explained in Section 2-3-2

1-5 Problem statement

This section outlines the problem statement starting with the research question and sub-questions, followed by the scope of the thesis.

1-5-1 Research question & sub-questions

As mentioned there are multiple societal benefits of people travelling by train as opposed to travelling by car. Unfortunately due to bad experiences (delays), people can be deterred from travelling by train. This thesis aims to reduce the negative experience by providing the TDs with a tool that can help reduce passenger delay. The TDs are time constraints when making decisions. This is the reason for using TADs since the decisions are predetermined and can be implemented directly. To compete with the TADs, computation time has to be limited.

Therefore the goal of this thesis is to answer the following main question:

Is it possible to develop a delay management method which
accurately reroutes passengers and still computes in a reasonable time?

This question can be split into the following sub-questions:

- *What is the currently available literature lacking?*
- *How to route and reroute passengers through a network?*
- *How to combine passenger rerouting and delay management?*
- *Is the proposed approach an improvement on the current approaches?*

These questions are still broad. The following subsection, the scope, narrows this down to be a comprehensible problem.

1-5-2 Scope of the thesis

Railway management is a broad concept with a lot of problems, rules and approaches to consider. This section states the restrictions and their impact on this thesis. In further research, the scope can be broadened. Therefore restrictions provide references to literature from subjects outside the scope.

Reduce passenger delay As stated the aim of the thesis is to reduce negative experiences as a result of delays. This thesis therefore will look into reducing the passenger delay. There are other options to reduce the negative effect of waiting on the environment. For more information on this topic, the reader is referred to [30].

This thesis will take the single-objective approach, not taking into account other operations like the management of crew or rolling stock. For a review of all railway operations, the reader is referred to [19].

Operational level The effects of delays on the passengers' journey can be reduced from a strategic level, a tactical level or an operational level. The strategic level concerns itself with network planning; location of the station and the tracks connecting the stations. The tactical level contains the scheduling of the timetable; passenger delay can be reduced by making the timetable more robust. On the operational level, passenger delay is reduced by TDs decisions, real-time managing the effect of delays. This thesis approaches passenger delay on an operational level. For literature on the tactical levels the reader is referred to [21]; a survey of robust timetabling.

No disruption In literature passengers' journeys are delayed as a result of either a delay or a disruption. Disruption is the temporary breakdown of (part of) the railway system as a result of train failure or a blocked train track. Disruption management includes the rescheduling of the rolling stock and crew. This thesis assumes disruptions do not occur. For an overview of disruption management, the reader is referred to [6] [14].

Macroscopic approach The DM problem can be approached from a macroscopic or microscopic point of view. Macroscopic models are roughly sketched. In general macroscopic topics consist of stations, tracks, departure and arrival times. In a more microscopic approach stations are divided into platforms and tracks into block sections. A microscopic approach is used more often in RTR which focuses on minimizing train delay, with a high focus on the feasibility of the network. The macroscopic model uses fewer data and therefore computes faster but still computes feasible solutions [16]. Since TD decisions require fast decision making, the majority of related work uses macroscopic railway models [19]. Therefore, the rest of this work uses the macroscopic railway model approach. The reader is referred to [11] for more information on RTR.

Full information All delays are assumed to be known such that decisions can be made with full information. In literature referred to as deterministic complete models. For literature, on stochastic or incomplete models the reader is referred to [19].

Smart routing, no capacity or spill Since this thesis is a proof of concept for passenger rerouting the problem is first run with the basic model of passenger routing, assuming the following:

- **Smart routing:** The assumption is made that passengers always have full information about the network and travel using the shortest path available.
- **No capacity:** Capacity constraints on trains [20].
- **No spill:** Passenger possibility to use another form of transportation when rerouting when delays are too severe, further explained in Section 1-3 and [20].

1-6 Thesis outline

This section outlines the structure of the thesis. A schematic overview is also shown in Figure 1-2. It is important to note that the structure of this thesis does not follow the chronological research approach. Chapter 2 and part of chapter 3 were subtracted from the literature review. Chapter 5 starts with the proposed new delay model which is computed using the offline (re)routing algorithm proposed in Chapter 4. Chapter 6 will test the proposed model against benchmark alternatives. Finally, Chapter 7 will evaluate the results; derive conclusions, state contributions and list possibilities for future work.

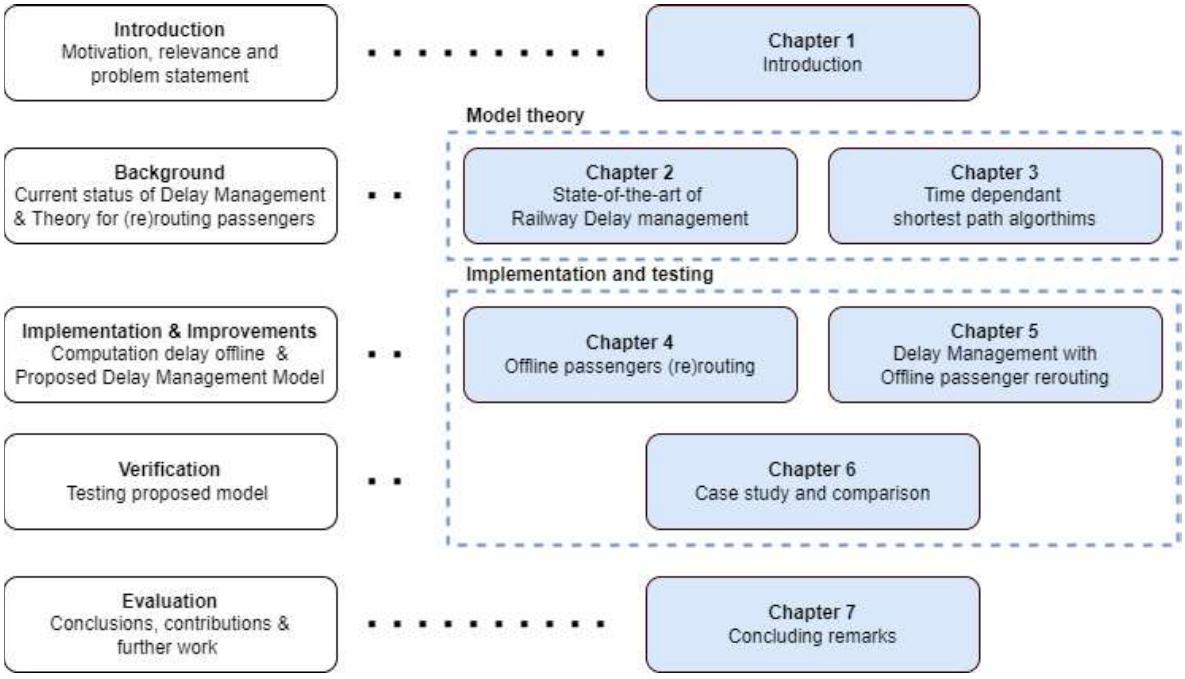


Figure 1-2: Schematic overview of thesis outline

Chapter 2

State-of-the-art of Railway Delay Management

An overview of the state-of-the-art in the field of railway delay management is given in this chapter. For a better understanding of the problem and to answer sub-question 1:

What is the currently available literature lacking?

This section has the following structure, starting with Section 2-1-1 on the railway systems definition, feasibility rules and Train Dispatcher input. Followed by Section 1-3 that models the Delay Management problem with Mixed Integer Linear Programming problem. Section 2-3 is an overview of relevant models for passenger focused delay management. Finally, Section 2-4 states the conclusions of this chapter.

2-1 Railway systems

This section is on the background of railway systems. The section starts with the railway definitions used in delay management, followed by the railway feasibility rules. The section is concluded with the inputs available to the Train Dispatcher (TD) to manage the effect of a delayed train. These terms and rules are the standards in current literature.

2-1-1 Definitions

This section is a summary of definitions used in the field of Railway management. A railway network is described by a set of stations and junctions (nodes) connected by tracks (edges). There can be parallel tracks between two nodes. Therefore the network can be described using a multi-graph. The trains in the network only stop at stations, where passengers can get on or off the train. Junctions are used for switching between tracks; the trains do not stop at junctions. The trains follow the routes and arrival and departure times from the

timetable. In general this timetable consists of the different train lines which repeat themselves throughout the day. A train line is a route from an origin station to a destination station with stops at stations on the route. Then after a short wait period at the destination station, it traverses the route in reverse. A train line can be dissected into multiple train runs; from one station/junction to another with no other station/node in between. The departure and arrival time of all the train runs is stated in the timetable. When delays occur the scheduled timetable is no longer possible and is therefore updated to a so-called disposition-timetable. The assumption is that disruptions do not occur. Therefore only the time of planned events needs to be adjusted. This disposition timetable needs to hold itself to the railway rules to maintain a feasible network.

2-1-2 Feasibility rules

As stated in the previous section, the disposition-timetable has to uphold the railway rules to remain feasible. The following feasibility rules apply:

- **Timetable:**; The actual departure and arrival time cannot be before the time for which it's scheduled.
- **Run time:**; A train run has a departure and an arrival event. The run time is the minimum time between the two. Each track has its own running time which can differ for different types of trains.
- **Dwell time:**; The minimum time a train has to wait at a station before departing. Giving passengers and crew the time to change. This also includes trains turning or changing lines at their end station.
- **Headway:**; The minimum time between trains travelling the same track, to avoid collisions.[1]

2-1-3 Inputs railway network

To minimize the effect of a delayed train the following inputs are available to the TDs:

Ordering: Two train runs traversing the same track have a scheduled order. However, the choice can be made to switch the ordering of the two trains.

Maintaining connection: In the case of a delayed feeder train the choice can be made to maintain the connection by delaying the departure of the connecting train.

[1] This more microscopic capacity constraint is introduced in [27].

Disposition-timetable: Contains the updated arrival and departure times of train runs. This timetable has to uphold all the railway feasibility rules. The initial delay can result in multiple updated timetable times as a result of the ordering, maintaining of a connection and the feasibility rules.

The combination of the ordering and maintaining of connections is referred to as the TD decisions. The TDdecisions and disposition timetable impact the journeys of passengers. The next section is on minimizing the effect of a delayed train on the total passenger delay using above mentioned inputs.

2-2 Delay Management as a Mixed Integer Linear Programming

As stated in section 1-3, Delay Management is used to minimize the effect of delays on the total passenger delay. This section starts with the definition of total passenger delay, followed by subsection 2-2-2 on the optimization method Mixed Integer Linear Programming (MILP). This section is concluded with subsection 2-2-3 on modelling he MILP as an Event Activity Network.

2-2-1 Total passenger delay

The total passenger delay is used as an indicator for the effect of the delay and TD decisions on all passengers. This total passenger delay can be described with the following general function:

$$\begin{aligned} &\min_{x,y} J(x,y) \\ &J(x,y) = \sum_{p \in \mathcal{P}} w_p * d_p(x,y) \end{aligned} \tag{2-1}$$

where J is the objective function, x the set of updated train run arrival and departure times and y the set of binary-decision variables. $\mathcal{P}$ is the set of all passenger Origin-Destination (OD) pairs, see paragraph below. w_p is the number of passengers of OD pair p, and $d_p(x,y)$ is the delay function of OD pair p with input variables (x,y).
The decision variables, $y(= y_{hw} \cup y_{wd})$, are the set of binary variables of the headway and the transfer constraints. The decision variables in the headway set, y_{hw}, state the order of trains, this is further explained in subsection 2-2-3. (which train goes first). The other, the wait-depart decision in the set, y_{wd}, whether a connection is maintained, this is further explained in subsection 2-2-3.

The formulation of $d_p(x,y)$ differs between different models in Delay Management (DM) literature. This is further discussed in Section 2-3.

Origin-Destination pairs: Passenger flow data is represented as OD pairs. OD pairs are denoted by origin station, destination station, the number of passengers and the arrival time at the origin station. Passengers are assumed to take the shortest path (travel time) to arrive at the destination station. The set of all OD pairs referred to as $\mathcal{P}$.

2-2-2 Optimization: Mixed Integer Linear Programming

In literature it is the common practice to model the DM as a Mixed Integer Linear Programming (MILP) problem [19]. MILP minimizes objective function (the summed passenger delay, eq. 2-1) while the constraints ensure a feasible network. The variables are the TD decisions and the disposition timetable, see Section 2-1-3. Mixed Integer Linear Programming (MILP) is defined as:

$$\begin{aligned} \min_{x,y} \quad & J(x,y) \\ J(x,y) = \; & Cx + Dy \\ \text{s.t.} \quad & Ax + By \geq b \\ & x \in \mathbb{R}^{|x|\times 1} \\ & y \in \mathbb{Z}^{|y|\times 1} \end{aligned} \tag{2-2}$$

Where $J(x,y)$ is the objective function that represents the goal of the optimization. x is the vector of real-valued variables, y is the vector of integer variables. If $J(x,y)$ cannot be represented as $Cx + Dy$ the problem is called Mixed Integer Non-Linear Programming (MINLP). The constraints have to be formulated in the form of $Ax + By \geq b$

The objective function $J(x,y)$ is linear as a result of $Cx + Dy$, if the objective cannot be formulated in this form the problem has to be solved with MINLP. Both are solvable but MILP is preferred over MINLP because the linear objective function results in a lower computation time.

Solving: The Mathematical Program Gurobi is used to solve the MILP optimization problem. This is proven to be the best solver for this kind of delay management problem [17].

2-2-3 Event activity network and constraints

It is the common practice in DM to translate the railway network with the railway rules into a Event Activity Network (EAN)[2] [19]. A EAN is a directed graph $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ where $\mathcal{E}$ is the set of events (nodes) and $\mathcal{A}$ the set activities linking the events. Each train run has two events; an arrival and a departure event, located at the origin and destination station respectively. This results in multiple nodes at each station. The set of all events is the combination of the set of all arrival events and all departure events:

$$\mathcal{E} = \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}} \tag{2-3}$$

As stated in section 2-1-1, the arrival and departure events E_{arr} and E_{dep} are scheduled with Π^a &Π^d. The actual arrival and departure times of the events are x^a & x^d respectively. They combine to $x(= x^a \cup x^d)$; the disposition timetable. The disposition timetable needs to uphold the timetable, run and dwell time constraints stated in subsection 2-1-2. This is represented

[2] also referred to as a time expanded model

as:

$$x_i \geq \Pi_i \qquad \forall i \in \mathcal{E} \tag{2-4a}$$
$$x_j \geq x_i + \tau_a \qquad \forall a = (i,j) \in \mathcal{A}_{\text{run}} \cup \mathcal{A}_{\text{dwell}} \tag{2-4b}$$

Where Π_i is the scheduled time of event i, τ_a is the minimum time between the two events of activity a. $\mathcal{A}_{\text{run}}$ is the set of all running activities and $\mathcal{A}_{\text{dwell}}$ is the set off all dwell activities.

The transfer and headway constraints can be represented in the following equations:

$$x_j \geq x_i + \tau_a + \beta y_a \qquad \forall a = (i,j) \in \mathcal{A}_{\text{transfer}} \cup \mathcal{A}_{\text{headway}} \tag{2-5a}$$
$$x_i \geq x_j + \tau_a + \beta(1 - y_a) \qquad \forall a = (i,j) \in \mathcal{A}_{\text{headway}} \tag{2-5b}$$

Where τ_a is the minimum time between the two events of a, β is a sufficiently large negative number such that the constraint always holds true if the binary variable $y_a = 1$. $\mathcal{A}_{\text{transfer}}$ and$\mathcal{A}_{\text{headway}}$ are the set of all transfer activities and the set of all headway constraints respectively.
If y_a is zero if the connection is maintained; ensuring that x_j takes place after $x_i + \tau_a$. r y_a is one if the connection is broken; the large negative number β results in x_j and x_i no longer constraining each other (the constraint always holds). This constraint is first introduced in [26]
The headway requires minimum distance between arrival and departure events of trains using the same line. Changing the order requires the constraints to be reversed. This is accomplished by switching between constraints. If y_a is zero the original constraint, equation 2-5a. If y_a is one the order is reversed , equation 2-5b.

Figure 2-1 shows a small example of an EAN with three passenger paths in different colours. This figure does not include the headway constraint. The scheduled timetable and all running-, dwell-, headway- and transfer times are known and constant.

Remark. *The timetable is scheduled with slack time; "extra" time between two events by which a delay can be reduced.*

2-3 Relevant Delay Management models

There are currently two main Delay Management approaches with regard to passenger rerouting: the basic and accurate approach. There is also a more theoretical approach that uses offline rerouting problems. The first part of this section is on the basic rerouting, the second part is on the accurate rerouting. The last section is the offline approach.

Differences in the approaches will be in the formulation of $d_p(x,y)$ (equation 2-1)and the addition of new variables and constraints.

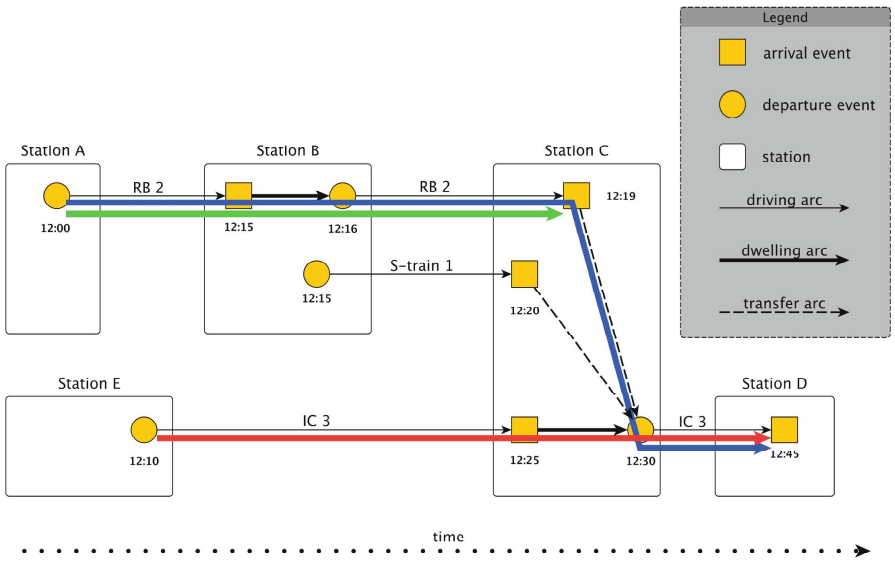


Figure 2-1: Small excerpt of an event-activity network with passenger paths [22]

2-3-1 Basic rerouting

The basic approach used in most of the literature was introduced by Schöbel [2001] [26]. The shortest paths for the OD pairs in $\mathcal{P}$ are assumed to be known. This set $\mathcal{P}$ can be split into two sets: direct OD pairs (with no transfer) $\mathcal{P}_D$ and connected OD pairs (with a transfer) $\mathcal{P}_C$. The delay of passengers of the $\mathcal{P}_D$ set is defined as the delay experienced at the final arrival event. The delay of a passenger of the $\mathcal{P}_C$ is defined as either:

Case $y_p = 0$**:** The connection on path p is maintained. The delay experienced by passengers w_p of path p is the delay of the train at the destination station x_p, similar to the delay of passengers of set $\mathcal{P}_D$.

Case $y_p = 1$**:** The connection on path p is broken, resulting in the passengers having to take the next connecting train. The connecting train on path p has a periodicity T_p. For simplification, this T_p will be the delay in case of a broken connection.

This is also referred to as the wait-or-depart decision and translates into the following objective:

$$J_{Basic,1}(x,y) = \sum_{p \in P_D} w_p(x_p - \Pi_p) + \sum_{p \in P_C} w_p((1-y_p)(x_p - \Pi_p) - + y_p T_p) \tag{2-6}$$

where P is the set of all OD pairs , w_p is the number of passenger travelling OD pair p, y_p is the binary connection variable, Π_p the scheduled arrival time of OD pair p, T_p is the periodicity of connecting train on OD pair p, x_p is the arrival event at the destination station of the shortest path of OD pair p if the connection is not broken.

An observant reader will have noticed that equation 2-6 would result in a MINLP problem. By replacing $(1 - y_p)x_p$ with a new variable q_p and introducing two new constraints, this can be rewritten in a MILP:

$$\begin{aligned} J_{Basic,2} = \sum_{p \in \mathcal{P}_D} w_p(x_p - \Pi_p) + \sum_{p \in P_C} w_p(q_p - \Pi_p(1 - y_p) + y_p T_p) \\ q_p \geq x_p - M y_p \quad \forall p \in \mathcal{P}_C \\ q_p \geq 0 \quad \forall p \in \mathcal{P}_C \end{aligned} \tag{2-7}$$

Where M is a large number; much greater than any time point in system that can occur in the system. With minimization of q_p this would result in $q_p = (1 - y_p)x_p$. The proof is given in [26].

Importance: The basic approach is the first mathematical model for passenger focused railway management. The model introduce for each OD pair with a transferrin $\mathcal{P}_C$ one binary variable and two constraints. The delay of passengers whose connection is broken is assumed to be the periodicity of the connecting train.

Remark: *Some literature uses the event delay instead of the event time as a variable. This would result in different modelling. In that case, M would be the largest allowed delay.*

2-3-2 Accurate rerouting

Accurate DM with the online rerouting of passengers is introduced by Berger [2011] [4], later improved upon by Dollevoet [2012] [12] and further improved by König [2021][20]. Online rerouting means that minimization of the total passenger delay and rerouting of passengers is done in the same optimization process by introducing a large number of new variables and constraints. New nodes are added to the EAN network: an origin event $\mathcal{E}_{org}$ and a destination event, $\mathcal{E}_{dest}$, at each station. These nodes are the start and stop event sfor routing the passengers. The (re)routing and minimization of passenger delay are formulated as:

$$J_{Online,1}(x, y) = \sum_{p \in \mathcal{P}_D} w_p(x_p - \Pi_p) + \sum_{p \in \mathcal{P}_C} w_p(x_p^* - \Pi_p) \tag{2-8}$$

$$q_{ap} \leq z_a \qquad \forall p \in \mathcal{P}, a \in \mathscr{A}_{\text{transfer}} \tag{2-9a}$$

$$\sum_{a \in \delta^{\text{out}}(e)} q_{ap} = 1 \qquad \forall e = \text{Org}(p) \in \mathscr{E}_{\text{org}} \tag{2-9b}$$

$$\sum_{a \in \delta^{\text{out}}(e)} q_{ap} - \sum_{a \in \delta^{\text{in}}(e)} q_{ap} = 0 \qquad \forall p \in \mathcal{P}, e \in \mathscr{E}_{\text{arr}} \cup \mathscr{E}_{\text{dep}} \tag{2-9c}$$

$$\sum_{a \in \delta^{\text{in}}(e)} q_{ap} = 1 \qquad \forall e = \text{Dest}(p) \in \mathscr{E}_{\text{dest}} \tag{2-9d}$$

$$x_p^* - x_e + M_2(1 - q_{ap}) \geq 0 \qquad \forall e = \text{Dest}(p) \in \mathscr{E}_{\text{dest}}, a \in \delta^{\text{in}}(e) \tag{2-9e}$$

Equation 2-8 is the objective which minimizes the average passenger delay. Where $\mathcal{P}_\mathcal{D}$and$\mathcal{P}_\mathcal{C}$ are the OD pairs without and with a connection respectively. x_p is the time of the arrival

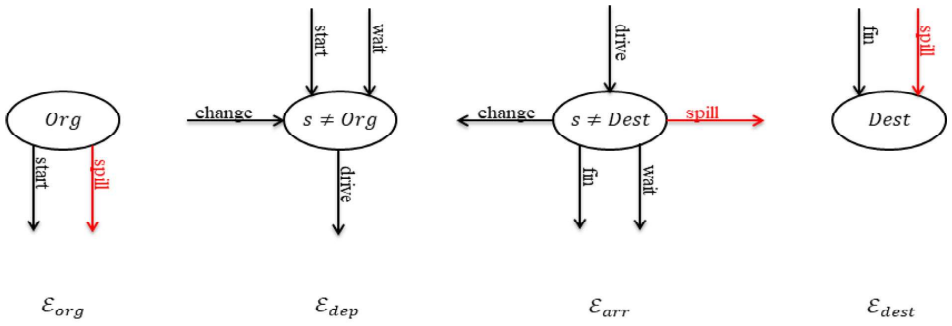


Figure 2-2: passengers with rerouting

event of a path without a connection, x_p^* is the destination event which can change when the connections are broken. q_{ap} is a binary variable whether an event a is used by passengers of OD pair p or not. $q_{ap} = 1$ in case the event is used, $q_{ap} = 0$ in case the event is not used. Equation 2-9a states that a transfer cannot be used if the connection z_a is broken; $z_a = 0$. Equation 2-9b makes sure only one arc/activity is used out of the origin event. Equation 2-9c makes sure that each event has the same number in-going as out-going activities used by passenger $p \in \mathcal{P}$. Equation 2-9d makes sure there is only one arc destination node at the final station. Equation 2-9e results in x_p^* having the same value as x_e: the arrival event at the destination station, after which the passengers go to the destination event at the destination station.

Train capacity with online rerouting is introduced by König 2021 [20] with the use of passenger fractions, turning q_{ap} from a binary variable into a real variable. Equation 2-9e does no longer hold with fractioning the passenger flow. This equation is therefore removed from the optimization and compensated for with a new objective 2-10a. Train capacity is ensured with equation 2-10b.

$$J_{Online,2}(x,y) = \sum_{p \in \mathcal{P}_D} w_p(x_p - \Pi_p) + \sum_{p \in \mathcal{P}_c} w_p \left[\sum_{a=(e,e') \in, A_{fin}(p)} x_e y_{ap} - \Pi_p \right] \tag{2-10a}$$

$$\sum_{p \in \mathcal{P}} w_p y_p \leq C_a \quad \forall a \in \mathcal{A}_{run} \tag{2-10b}$$

Where $\mathcal{A}_{\text{fin}}(p)$ is the set of arcs liking arrival events at the destination station of p with the destination event.

Importance: The accurate online rerouting of passengers performs better in minimizing the total passenger delay compared to the basic approach. However, the computation time of the offline approach is far worse due to the addition of a larger number of binary variables and constraints[3] in comparison to the basic approach[12].

[3]It is not easy to quantify the number of added variables and constraints, it is highly dependent on the scale the network and the lengths of the passenger paths.

Offline rerouting

Previously stated objectives either assumed a delay to be T_p or reroutes passengers with an online algorithm. The former has fast computation the latter has better accuracy. Cavone [2020] [7] proposes an improvement on the basic approach by using a different arrival event in case the connection is broken. The arrival event is computed before the optimization for reducing the passenger delay. This approach is therefore referred to as the offline approach. The objective of the offline approach is formulated as:

$$J_{\text{Offline},1}(x,y) = \sum_{p\in P_D} w_p(x_p - \Pi_p) + \sum_{p\in P_C} w_p((x_p^{or} - \Pi_p)(1-y_p) + (x_p^{al} - \Pi_p)y_p) \tag{2-11}$$

where x_p^{or} is the arrival event of path p in case the connection is maintained, x_p^{al} is the arrival event of path p in case the connection is broken and P_D and P_C are set of direct paths (no connection) and set of connection paths respectively.
This objective can be linearized to a MILP with the same principle of equation 2-7. By replacing $(1-y_p)x_p^{or}$ and $y_p x_p^{al}$ with q_p^{or} and q_p^{al} respectively, in combination with introducing four constraints:

$$\begin{aligned} J_{\text{Offline},2}(x,y) = \sum_{p\in P_D} w_p x_p + \sum_{p\in P_C} w_p(q_p^{or} + q_p^{al}) \\ q_p^{or} \geq x_p^{or} - \beta y_p \quad \forall p \in P_C \\ q_p^{or} \geq 0 \quad \forall p \in P_C \\ q_p^{al} \geq x_p^{al} - \beta(1-y_p) \quad \forall p \in P_C \\ q_p^{al} \geq 0 \quad \forall p \in P_C \end{aligned} \tag{2-12}$$

where β is a large number in order to neglect the corresponding constraint if necessary.

Importance: The proposed offline approach is only tested on a small model to prove its effectiveness. However, the alternative shortest events were computed by hand. This method introduces 2 binary variables and 4 constraints for each $p \in \mathcal{P}_C$.

2-4 Conclusion

Online rerouting is accurate but has the disadvantage of a large increase in computation time. The thesis aims to be applicable in a Decision Support System (DSS) low computation time is a requirement, to provide the TDs with a tool that is useful during operation. Offline rerouting fulfils these requirements by computing alternative routes before the delay management optimization. The paper of Cavone 2020 [7] tests this principle on a small network where the shortest path and alternative shortest path can easily be computed by hand.

Pathfinding algorithms that compute routes and alternative routes before the delay management problem will provide a flexible method to be easily applied to larger networks. The next chapter is on the theory required for the formulation of such an algorithm.

Chapter 3

Time dependant shortest path algorithms

The previous chapter states the need for a pathfinding algorithm that can route and reroute passengers before the Delay Management (DM) optimization, also referred to as offline rerouting. This chapter lays the foundation required to answer the sub-question:

How to route and reroute passengers through a network?

Section 3-1 states the requirements of the path finding algorithm. Section 3-2 explains the time-dependant graphs. The third Section 3-3 is on Dijkstra's algorithm. Finally, Section 3-4 concludes this chapter with a summary.

3-1 Requirements of path finding algorithm

The requirements for the pathfinding algorithm to be used for Delay Management (DM) with offline (re)routing are the following:

- **Time-dependent edges** In contrast to most pathfinding problems railway systems have time-dependent edges; the edge weight changes over time.
- **Time complete** Since the edge weight can vary depending on the departure time consequently the best path/route between stations can vary depending on departure time. Therefore the output of the pathfinding algorithm needs to provide a solution for any departure time in a given time interval.
- **Large Network** The algorithm should be able to handle large networks to be applicable as a Decision Support System (DSS).

- **Earliest Arrival** As mentioned in 2-2-1, passengers are assumed to travel using the shortest path; the path with the earliest arrival time. The arrival event at the final station is defined as x_p. This is calculated with a Shortest Path Algorithm (SPA).
- **Path retrieval** Path retrieval is required to list paths containing a connection and to have the ability to model passenger flow.
- **Alternative shortest path** The output of the algorithm should also provide an alternative path for the paths that contain a connection.
- **All-pair** The algorithm needs to provide shortest and alternative shortest paths between all possible pairs of stations. In graph theory, this is called an All-Pair Shortest Path Algorithm (APSPA) problem.

The following section reviews literature which is applied in the pathfinding algorithm in section 3.3.

3-2 Time-dependant graphs

Graphs are the mathematical structure used for shortest path algorithms. This section first gives a short introduction to graph theory, followed by Section 3-2-2 on time-dependant vs time-expanded graphs. Finally, Section 3-2-3 is on piecewise linear functions as time-dependent edge weights.

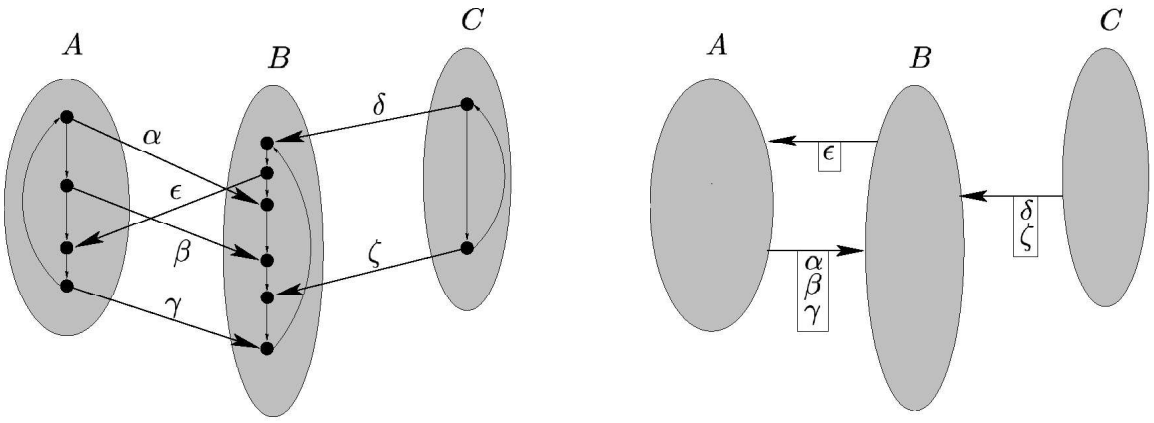
3-2-1 Graph theory

The basics of the graph are as follows: a graph is a system modelled with a set of nodes V and a set of edges E. An edge connects two nodes $e = (u, v),\ e \in E\ \&\ (u, v) \in V$. When the edge is directed u is called the tail and v is called the head of the edge e. The set of nodes and edges combined in a tuple graph $G = (V, E)$ is called a graph. When all edges E are directed, G is called a directed graph. In most applications, the weights of the edges are time-independent (constant).

3-2-2 Time-Dependant and Time-Expended Model

There are two baseline approaches for modelling public transportation: the Time-Dependant and the Time-Expended model. The online rerouting applies the time-expended approach. This approach uses multiple nodes at each station: one for each arrival- and departure event at the station, with directed edges (activities) connecting the events, figure 3-1a. In the time-dependent approach, nodes are stations[1] or junctions and tracks with multiple train run labels for edges, figure 3-1b [12]& [20]. The time-dependent approach far outperforms the time-expended model approach for large networks [23], [24] & [3]. This is the effect of the time-dependent approach requiring fewer nodes and edges in comparison to the time-expanded approach, see figure 3-1. For this reason, the rest of this work uses the time-dependent approach for pathfinding.

[1]The station can be split into platforms in a more microscopic approach.



(a) Time-expanded digraph

(b) Time-dependent digraph

Figure 3-1: Comparison of time-expanded and time-dependent digraph of the same timetable [24]. With stations A,B,C and train runs: α, β, γ, δ, ϵ & ζ

3-2-3 Time-Dependant edges: piecewise linear function

The edge weight in a time-dependant network is not constant, as in basic graph theory, but a Travel Time Function (TTF) f. The TTF $f_{i,j}$ specifies the time $f(\tau)$ needed to go from station i to station j with departure time τ. Piece-wise linear functions can be used as TTF. There are two types of piecewise linear functions: interpolation and travel time plus weight time, figure 3-2. A point is represented as tuple (a, c) where a is the departure time and c[2] the travel time[3]. For railway networks travel time plus weight time is used.

Shortest path algorithms require edges to be evaluated, linked and merged. These are evaluated in this section in this order. The next paragraph is on the First In First Out (FIFO) property: a requirement of the set of TTF for most pathfinding algorithms. This section is concluded with a paragraph discussing the grading complexity of the TTFs.

Evaluating Evaluation is the edge cost $f_{i,j}(\tau)$ at time τ, the (wait till discrete event) piecewise linear function $f_{i,j}(\tau)$ uses a set of points for this evaluation. The set is ordered by departure time. For evaluation of cost at time τ the earliest departure after τ is selected. The cost at time τ is the time until this point summed with the travel time c of this point.

$$f_{i,j}(\tau) = (a - \tau) + c \tag{3-1}$$

Linking Linking is the combing of two TTFs of adjacent edges into one. Linking two edge cost $f_{i,j}$ (from node i to node j) with the cost $f_{j,k}$ (from node i to node j) creating $f_{j,k}$. This can be accomplished by creating in $f_{i,k}$. $f_{i,k}$ is generated by creating new point by combining the iterating trough all points x in $f_{i,j}$ en appling the following equation:

$$c(i,k) = c_{i,j}(x) + f_{j,k}(a_{i,j}(x) + c_{i,j}(x)) \tag{3-2}$$

[2]Most pathfinding algorithms require non-negative edge weights; $\tau \leq \tau + f(\tau)$

[3]The required information of a point to compute the travel time are the departure and the travel time. Additional information can be added to the points. Information like; the previous station, train line etc.

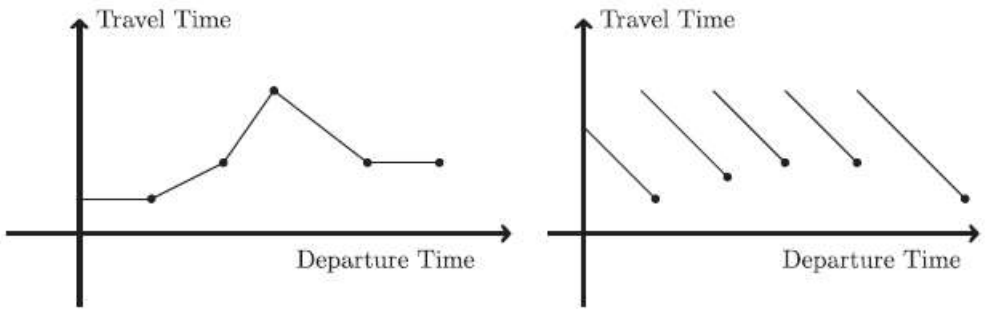


Figure 3-2: Two examples of piecewise linear travel time functions. On the left interpolation, used for roads. The function on the right is travel time plus weight time, used for railway networks.[13]

Alternatively the same can be accomplished by function composition. This is defined by $f_{i,j} \oplus f_{j,k} := f_{j,k} \circ f_{i,j}$ resulting in $f_{j,k}(f_{i,j}(\tau))$.

Merging Merging is combining two TTFs of parallel edges $f_{i,j}, f'_{i,j}$ into one TTF $f_{i,j}$, removing points which would not be used since the following point would result in an earlier arrival time. Merging is defined by $\min f_{i,j}, f'_{i,j}$ with $\min f_{i,j}, f'_{i,j}(\tau) := \min f_{i,j}(\tau), f'_{i,j}(\tau), \tau \in \Pi$ where Π is the time interval. Alternatively merging both sets of points in a new ordered list $f^*_{i,j}$. Then iterating trough all points; a point x is removed if the travailing using the following point $x + 1$ would result in an earlier arrival time:

$$a(x) + c(x) > c(x + 1) + a(x + 1) \tag{3-3}$$

The merging of two parallel edges is the removal of points that are redundant since a later departure point would result in an earlier arrival time. Applying the merging principle results in merged edges upholding the FIFO property.

FIFO property The FIFO property means that a train cannot be overtaken by a train (travelling on the same edge) that departs at a later time. This is also referred to as the the non-overtaking property or the function being monotonic. The FIFO property is formulated as:

$$f_{i,j}(x) + x \leq f_{i,j}(y) + y \quad \forall x, y \in \Pi \text{ and } x \leq y \tag{3-4}$$

where Π is the interval of time.

Complexity The complexity of a piecewise function f is denoted by the number of points in the function $|f|$

3-2-4 Frequency labels

As stated in the previous paragraph, the complexity of a TTF is the number of labels within. The number of labels can be reduced by introducing frequency labels [3]. Labels can be compressed if they occur with a certain periodicity, which is common in public transportation.

The frequency labels are represented as tuple $([a, b], p, c)$ where a is the start time and b the end time of the interval of periodicity p is the period of the label and c is the cost of the label. The cost of frequency label f at time τ can be evaluated with the following formula:

$$f(\tau) = \begin{cases} a - \tau + c & \text{if } \tau < a \\ a + \lceil (\tau - a)/p \rceil \cdot p - \tau + c & \text{if } \tau \in [a, b] \\ \infty & \text{if } \tau > b \end{cases} \tag{3-5}$$

Linking and merging are further discussed in [3].

3-3 Dijkstra's algorithm

Modified Dijkstra is the most common algorithm for time dependant shortest path algorithms [9]. The first part of this section discusses Dijkstra for time-independent networks, followed by Section 3-3-2 on the modified Dijkstra algorithm for time-dependent networks.

3-3-1 Time independent Dijkstra

Dijkstra's algorithm [10] is the standard algorithm for the one-to-all shortest path problem for time-independent edge weights. This algorithm requires a directed graph with non-negative edge weights as input. The algorithm starts by selecting an origin node s and setting all distances to other nodes to infinity except the distance to itself $\text{Dist}[s, s] = 0$, finishing initialization with adding s and $\text{Dist}[s, s]$ to Q. Q is the priority queue that depicts the current search horizon around s. Each iteration step, the algorithm removes the node v from Q with the shortest distance from s. Then, all outgoing edges v, w from node v are scanned for edge relaxation;

$$\text{Dist}[s, v] + len(v, w) < \text{Dist}[s, w]$$

If the relaxation holds, a shorter path to w via v has been found, followed by either inserting node w in the priority queue or, updating the distance if node w is already in the priority queue.

Label setting: Dijkstra's algorithm has the label setting property: once a node v is extracted from Q, the $Dist[s, v]$ is the correct minimum distance. Therefore, if Dijkstra's algorithm is applied to a one-to-many problem the algorithm can stop after all nodes in the set of many are removed from Q.

Path retrieval: Path retrieval with Dijkstra can be accomplished by storing the previous station of a node. Similar to the distance in initialization all the previous stations are set to infinity. This previous station is updated each time an edge is relaxed. If edge (v, w) is relaxed, the previous station of w is set to v. Since each edge has a previous station, the path from s to w can be backtracked from w.

```
   Data: Source s & G = (V, E) Edge weights are non-negative.
   Result: Dist; List of shortest path labels & Prev; used for path retrieval
 1 %initialization;
 2 forall v ∈ V\s do
 3 |   Dist[s, v] ← inf
 4 |   Prev[s, v] ← inf
 5 end
 6 Dist[s, s] ← 0
 7 add [s, Dist[s, s]] to Q
 8 %main loop;
 9 while Q not empty do
10 |   v ← Node in Q with min Dist[s, v]
11 |   remove v from Q
12 |   forall (v, w) ∈ E do
13 |   |   alt ← Dist[v]+length(v, w)
14 |   |   if alt < Dist[w] then
15 |   |   |   Dist[w] ← alt
16 |   |   |   Prev[w] ← v
17 |   |   |   add [w, Dist[w]] to Q if w already in Q update label
18 |   |   end
19 |   end
20 end
```

Algorithm 1: Basic Dijkstra with path retrieval

3-3-2 Modified for time dependent network

Dijkstra can be modified to compute the all-pair shortest paths in a time-dependent network. The required input for this method is a time-dependent directed graph with non-negative FIFO edge weights. There are two forms of this modified Dijkstra algorithm with the difference in output: time-query or profile-query. The time-query computes the fastest path for a given departure time. The output is denoted as $d(s,t,\tau)$ for the distance of the shortest path between node $s,t \in V$ and departure time τ, where V is the set of all nodes. The profile-query computes a TTF of all best connections during a given time interval. The time-query is denoted as $d_*(s,t)$[9]. The time-query output is required to be time complete, see Section 3-1.

Time-query output Modified Dijkstra with a time-query output [15] $d(s,t,\tau)$ can be solved similar to basic Dijkstra. In this modification edges are relaxed if:

$$\text{Dist}[s,v,\tau] + \text{len}[v,w,\text{Dist}[s,v,\tau]] < \text{Dist}[s,w,\tau]$$

This method loses the label setting property as a result of the time-query output so once a node t is removed, the shortest path $d(s,t,\tau)$ is correct.

Time-profile output Modified Dijkstra with a time-profile output [8] $d_*(s,t)$ starts with initializing $d_*(s,s) \equiv 0$ and $d_*(s,u) \equiv \text{inf}$. Then, each iteration the node with minimum distance to s is selected, $\underline{d}_*(s,u)$. Then, for all outgoing edges form u a temporary label is created $l(v) = d_*(s,u) \oplus \text{len}_{s,v}$. $l(v)$ is an improvement of $d_*(s,v)$ if label $l(v) \geq d_*(s,v)$[4] does not hold. $d_*(s,v)$ is updated if $l(v)$ is an improvement, updating is the merging of the two labels $min\{l(v), d_*(s,v)\}$. Then, after updating $d_*(s,v)$, v is (re)inserted into the queue. In the previous methods reinstating was not needed. This is referred to as the label setting property. This method does not have this property, nodes can therefore be reinstated in the Q, this is referred to as the label-correcting approach. The algorithm is stopped if the queue is empty.

The use of time-query is limited: just computing the shortest path for a given time τ. The profile-query gives a better overview of all shortest paths, which is needed for the time complete requirement from Section 3-1. Therefore, the focus of this thesis is on profile-query output.

3-4 conclusion

A modification of the Dijkstra algorithm provides a suitable approach for the offline (re)routing of passengers in the railway network. This method can be applied to time-dependent edges. Will provide the time complete shortest paths for all origin-destination pairs in the system with the time-query output. Can process arbitrary large networks. Its output can also be used to compute alternative paths for routes with a connection. The implementation of this algorithm with the computation of the alternative routes is discussed in the following chapter.

[4] The label has higher distance for all $\tau \in \Pi$.

Chapter 4

Offline passenger (re)routing

The previous chapter laid the theoretical foundation for time-dependent shortest path algorithms. This theory is implemented in this chapter to propose an algorithm that finds all the shortest paths in an arbitrary large highly travelled railway network and also competes as an alternative route for shortest paths with a connection. This chapter will answer the sub-question:

How to route and reroute passengers through a network?

This chapter is structured with the steps required for computing the alternative paths, see Figure 4-1. The outline of this chapter is as follows: first Section 4-1 on the required timetable input information and the pre-processing of the timetable information. Next, Section 4-2 on the shortest path algorithm for the routing of passengers. Finally, section 4-3 is on the calculation of alternative paths.

4-1 Input algorithm

This section focuses on the input of the pathfinding algorithm. The required timetable information is the first part of this section; this will include the required timetable information for creating the constraint in the next chapter. Thereafter a subsection on the pre-processing of timetable data into a time-dependent graph, such that it is suitable as input for the pathfinding algorithm.

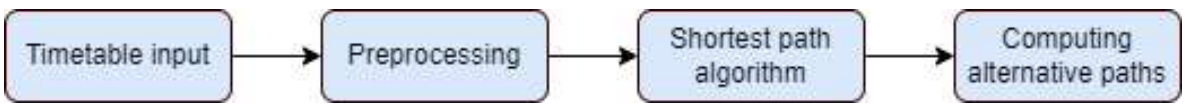


Figure 4-1: Steps for rerouting passengers

4-1-1 Timetable information

The timetable information required for the pathfinding algorithm and delay management should consist of the set V of train lines split into set I of unique train runs. In this thesis, the timetable is assumed to be cyclic with periodicity T_{Π}. This assumption makes it so that, only for departure times in the interval of $[0, T_{\Pi}]$ shortest paths have to be calculated. Departure times outside the interval are similar to the shortest path within the interval, only shifted with period T_{Π}.

Each train run should have the following information:

ID The each train run has a unique id for identification $i \in I$.

Train line Each train run is connected to a train line $v \in V$. The first train run of a train line is at a turning point, see Section 2-1-1.

Origin & destination node Each train run has an origin and a destination node. A node can be either a station or a junction, see Section 2-1-1.

Track There can be multiple tracks between nodes (see Section 2-1-1), tracks therefore also need an ID. Track information is required for forming the headway constraints, see Section 2-1-2.

Departure & arrival time Scheduled departure time from the origin node and scheduled arrival time at the destination node.

Run and dwell time Required minimum time between two linked nodes event, see Section 2-1-2.

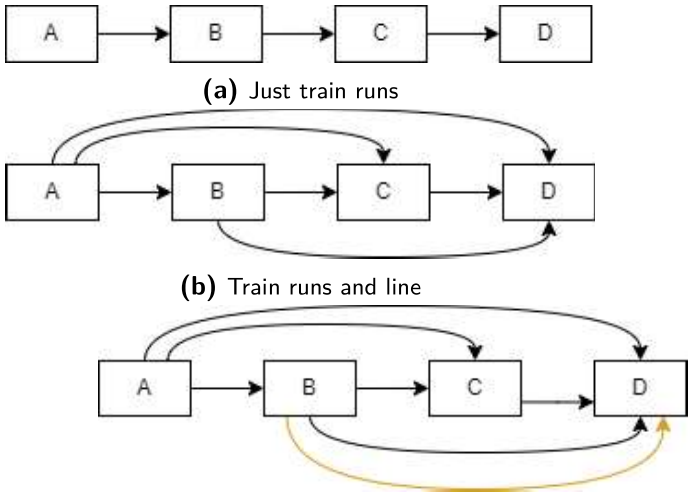
Shift Is a for ensuring the departure time to be between zero and T_{Π}. There is a shift of T_{Π} if the previous train run (of the same line) has an arrival time larger than T_{Π}. If a shift occurs the shift variable is set to one, this variable is otherwise zero,

Code: *The timetable is loaded into Matlab with the code from Appendix A-1.*

4-1-2 Preprocessing timetable information

The timetable has to be processed before it can be used as input for the algorithm. First, the timetable has to be reduced to only relevant information. The second step is transforming the timetable information into a time-dependent graph. Next train runs from the same line are connected. Finally, the resulting graph is made to uphold the First In First Out (FIFO) property.

Reduce timetable In the first-order approach in this thesis, the shortest paths are calculated under the assumption of no delays. The network is always feasible under this assumption. Junctions and stations where the train does not stop are irrelevant. The train runs with a dwell of zero are linked resulting in a new timetable with just the stations where passengers can board or get off a train.



(a) Just train runs

(b) Train runs and line

(c) Reason why line

Figure 4-2: Third step: make train line

Make graph The reduced timetable is transformed into a time-dependent graph by iterating through the train runs, turning them into frequency labels[1] and adding these frequency labels to the edges. The frequency labels are of the following structure: $[i, a, p, c, n_t, \text{prev}_s]$. Where i is the run id for identification, a is the start time, p the periodicity, c is the cost, n_t is the number of transfers and prev_s is the previous transfer station. In this thesis, the frequencies p of all labels are assumed to be T_Π. In the prepossessing phase, all frequency labels have zero for both n_t & prev_s because transfers are made in the pathfinding algorithm. These frequency labels are also under the assumption that the periodicity is constant and day-round. The cost of a frequency label at time τ can be evaluated with the following function:

$$f(\tau) = a + \lceil (\tau - a)/p \rceil \cdot p - \tau + c \tag{4-1}$$

Form train lines The third step is to connect stations of the same line with new edges; shown in Figure 4-2. Figure 4-2a shows a directed graph of one train line[2], Figure 4-2b shows the added edges as a result of merging runs of the same train line. This step is done to ensure that the output of the shortest path algorithm not only provides the shortest path but also takes into account paths with fewer transfers. Figure 4-2c show an example of the scenario where the fastest route from A to D is: using the black line to station B, then transferring to the orange line to station D. It is important to know that passengers have an alternative route which might take longer but has fewer transfers. The algorithm would not show the option of no transfer because it would have the same start time with a longer travelling time. By adding the train lines into G_{in} it is ensured that the train lines will show up in the output of the algorithm.

Make FIFO The fourth step is removing the redundant labels because alternative routes are faster. There are redundant labels since there are differences in train line speed and route.

[1] See Section 3-2-4

[2] the train line is in one direction for this example

When travelling long distances, a slow train will not be used if there is an alternative train that departs later with an earlier arrival time. The removal of these labels is accomplished by iterating through all edges and for each edge remove the non-FIFO labels; labels with similar departure time and later arrival time and labels with earlier departure time and similar or later arrival time.

The resulting graph is suitable for the input of the pathfinding algorithm.

Code: *The code for preprocessing the timetable data can be found in appendix A-2.*

4-2 Shortest path algorithm

The previous subsection was on how to translate the cyclic timetable into the input graph G_{in} of the algorithm. This section proposes a modified Dijkstra algorithm that computes the all-pair time-query shortest paths from the input graph G_{in}. The section starts with the schematic modified Dijkstra algorithm and is concluded with an explanation of the output of the algorithm.

4-2-1 Modified Dijkstra

The proposed shortest path algorithm is shown in Algorithm 2. This is still the basis of the time-dependent Dijkstra algorithm. Differentiation occurs is with the edge relaxation shown in algorithm 3. In the proposed algorithm edge relaxing, the updating of Dist$[s, w]$, is accomplished by combining the edge labels of $[s, v]$ and $[v, w]$ if any of these combinations is an improvement on the current edges-labels of $[s, w]$. The edge labels which are an improvement are added to $[s, w]$. Then, all the non-FIFO edges are removed with the following exception. In the case that label l_2 has the same start time as label l_1 but a larger cost it should be removed but it is not if l_2 has fewer transfers than label l_1. This feature results in that some shortest paths are not removed because passengers like to travel using fewer transfers.

4-2-2 Output algorithm

The output of the shortest path algorithm is G_{out}. This graph contains the time-profile shortest paths between all stations in the network. This information can be translated into OD-pairs, see section 2-1-1. The number of passengers of each OD pair can be generated with these OD pairs.

Code: *The code for shortest pathfinding is stated in appendix A-3.*

4-3 Alternative path

With the shortest path graph G_{out} of the network, the alternative routes for missing a transfer can be computed. For illustrating the alternative paths for passengers whose connection is

Data: G_{in}; input graph with stations for nodes and list of frequency-labels [3]for edges.
s; origin station.
Result: G_{out}; output graph containing all shortest-path frequency-labels from station s to all other station in the network.

```
/* initialization                                                    */
forall v ∈ V\s do
    Dist.min[s, v] ← inf
    Dist[s, v] ← []
end
Dist.min[s, s] ← 0
Q ← [Dist.min[s, s],s]
n.stations ← length(G_in)
G_out ←graph(n.stations)
/* main loop                                                         */
while Q not empty do
    v ← station from Q with min Dist.min[s, v]
    forall (v, w) ∈ E do
        if possible update Dist[s, w] with path [s, v, w]/* see algorithm 3   */
        if Dist[s, w] is updated: update Q with w
    end
    remove v from Q
end
```

Algorithm 2: One-to-all shortest path algorithm

Data: Dist$[s, w]$:distance (labels) between station s and w
Dist$[s, v]$:distance (labels) between station s and v
Dist$[v, w]$:distance (labels) between station v and w
Result: Dist$[s, w]$:distance (labels) between station s and w
added: which is True if a labels has been added to Dist$[s, w]$ otherwise False

```
added ← False
forall label-OC in Dist[s, v] do
    labels-CD ← labels from Dist[v, w] with of different train line label-OC
    label-CD← combine label-OC and labels-CD with taking transfer time into
      account
    if label-CD is improvement of Dist[s, w] then
        Add label-CD to Dist[s, w]
        added← True
    end
end
if added then
    make Dist[s, w] FIFO with fewer transfer exception
end
```

Algorithm 3: Try update distance label

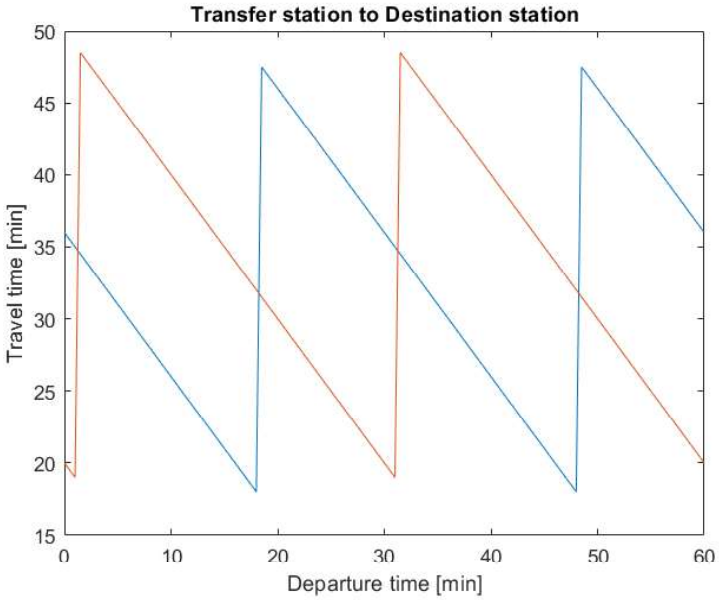
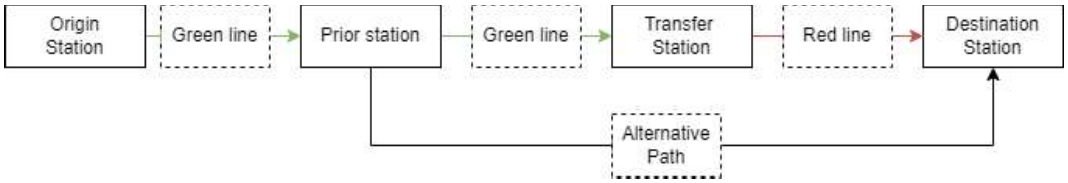


Figure 4-3: Two shortest paths of different train lines

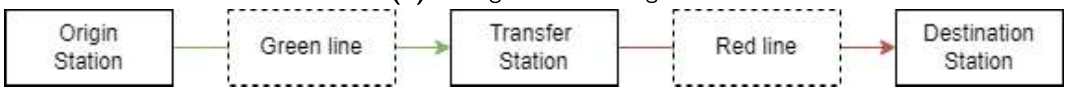
broken, the following scenario is used: a passengers shortest path was to take the green line, then transfer to the red line to reach the destination station (see Figure 4-4b). This section states the alternative paths if this connection is broken. The assumption is that passengers decide on which alternative route to take at the transfer station, alternative paths in the form of Figure 4-4a are not taken into account. This would complicate the objective and is therefore disregarded in this thesis. Three alternative path scenarios remain. The first scenario is to take the next train of the same line of the missed connection, figure 4-4b. This scenario is the common approach in literature [19]. The delay of this alternative route is the periodicity of the connecting train line. This is not always the fastest alternative, this thesis expands on this with the scenarios of a different line at the transfer station or a different path after remaining seated.

The scenario of the different alternative paths is shown in Figure 4-4c. The alternative routes appear in the time-profile output of the algorithm, Figure 4-3. It shows how the blue line results in a lower travel time when the connection to the red line is missed. This alternative route can be taken easily from the time-profile and would result in a delay of 17 minutes instead of 30 minutes.

The time-profile output (Figure 4-3) does not that show there is an alternative route faster route; from the next station on the green line. Since the time-profile output only shows the fastest route this alternative route is not retrievable from this time query. The solution is to simulate the passenger remaining seated in the green line. This path is checked by the following steps: travelling on the green line to the first station after the transfer station, from this station checking the time profile for the fastest path to the destination station, using this route if it results in a smaller delay than the other two alternatives.

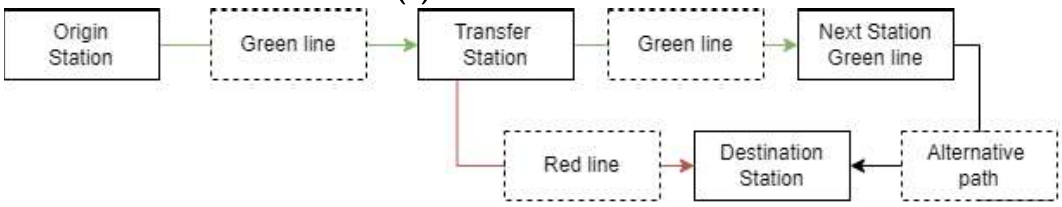


(a) Disregarded rerouting



(b) Wait frequency of connecting train

(c) Alternative train line



(d) Alternative route next station

Figure 4-4: Alternative routes

Delay of broken connection The delay θ_p of the passengers of OD-pair $p \in \mathcal{P}_C$ is computed by subtracting the arrival time of the original path from the arrival time of the alternative path. In the next chapter, this delay will be used to compute the cost of breaking a connection.

Code: *The code for computing the delay of the alternative path is stated in in appendix A-4.*

Chapter 5

Passenger delay management with offline passenger rerouting

This chapter proposes a Decision Support System (DSS) for a large highly travelled train network that minimizes passenger delay using offline passenger rerouting. A schematic overview of the DSS is shown in Figure 5-1. The previous chapter proposes an algorithm that fulfils numbered blocks 4 & 5 from Figure 5-1. This chapter will discuss the other blocks

The structuring of this chapter will use the numbers of Figure 5-1. The chapter starts with input for modelling the Mixed Integer Linear Programming (MILP): the timetable data and passenger flow data. Next Section 5-2, on the model; generating the constraints and stating the proposed objective of this thesis. The last section,5-3 , describes the Output of the DSS (9, 10 & 11).

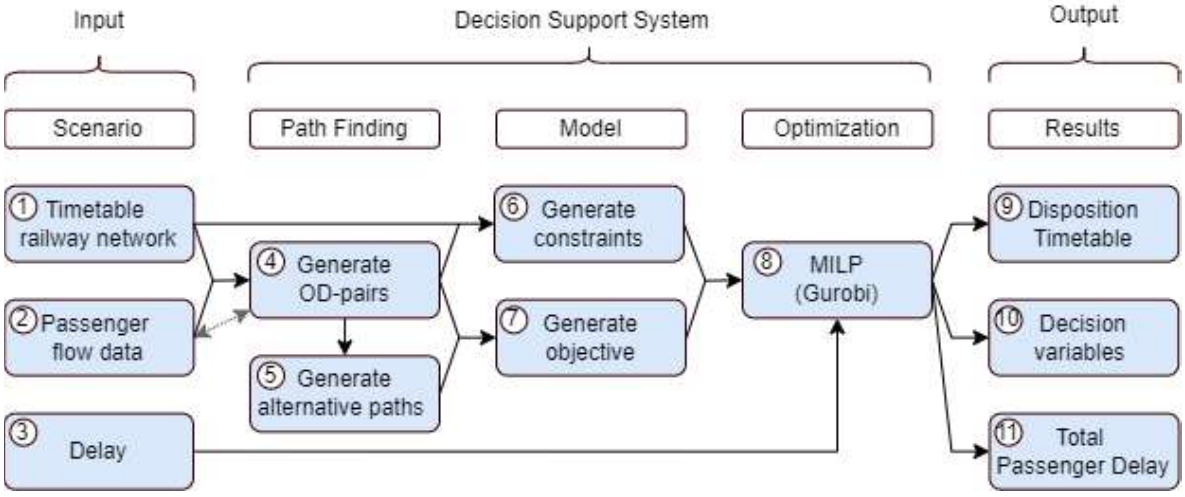


Figure 5-1: Schematic Overview proposed Decision Support System (DSS)

5-1 Input for Mixed Integer Linear Programming problem

This section is on the input for the MILP problem; the timetable and passenger data. First, the timetable data is discussed and how this can be transformed into a dayplan. Secondly, the passenger data is discussed.

Timetable to dayplan Section 4-1-1 already stated the required information for the timetable; id, line. origin, destination, departure- and arrival time, run- and dwell time and the shift. This timetable is assumed to have periodicity T_{Π}. As a result, the timetable can be repeated multiple times resulting in a dayplan with a span of a multiplicity of T_{Π}.

Passenger data Passenger data can be either provided or generated. If the passenger data is provided, it can be combined with the shortest paths to create the Origin-Destination (OD) pairs. However, if this data is not provided passenger data can be generated and coupled to the shortest path.

Code: *Appendix A-6 states the code for forming a dayplan.*

5-2 Delay Management Model

5-2-1 Constraints

The constraints restrict the optimization methods in altering the variables. The constraints are modelled with the theory of Section 2-2-3. The feasibility constraints can be subtracted from the dayplan which contains the required information; departure-, arrival-, run- and dwell time and the track id[1], see Section 4-1-1. The connections constraints can be formed with the output of the proposed shortest path algorithm and the unreduced timetable.

Introducing delay Delays are introduced by altering the timetable constraint. Altering the lower bound of arrival or departure event is similar to introducing a delay.

Code: *Appendix A-7 states the code for transforming the dayplan into a MILP . Appendix A-8 states the code for selecting relevant constraints and variables.*

5-2-2 Proposed objective

As stated in Section 2-3, the proposed object by Cavone [7] uses an alliterative arrival event. This thesis uses a first order approach for modelling the objective with the following structure:

$$J_1(x,y) = \sum_{p \in P_D} w_p(x_p - \Pi_p) + \sum_{p \in P_C} w_p((x_p - \Pi_p)(1 - y_p) + \theta_p y_p) \tag{5-1}$$

[1] A universal headway time τ_a for $a \in \mathcal{A}_{headway}$ is assumed to be known

where θ_p is the delay passenger of OD pair $p \in P_C$ in case their connection is broken. This value is computed with Section 4-3. y_p is the decisions variable if a connection is broken. Multiple variables y_p of different OD pairs in $\mathcal{P}_C$ refer to the same connection variable in $y_i \in y$.

This function is non-linear and can be linearized to the following equation with the addition of one variable q_p and two constraints per variable x_p with $p \in \mathcal{P}_C$:

$$\begin{aligned} J_2(x,y) = \sum_{p \in \mathcal{P}_D} w_p(x_p - \Pi_p) + \sum_{p \in P_C} w_p(q_p - \Pi_p(1-y_p) + \theta_p y_p) \\ q_p \geq x_p - M y_p \quad \forall p \in \mathcal{P}_C \\ q_p \geq 0 \qquad\qquad \forall p \in \mathcal{P}_C \end{aligned} \tag{5-2}$$

The introduced variable q_p with the two constraints is equal to $x_p(1-y_p)$ when the objective $J_2(x,y)$ is minimized.

This objective has the same number of variables and constraints as the basic rerouting approach, with a delay estimation τ_p which is similar or smaller compared to this group.

5-3 Output

The constraints and the objective from the MILP problem which is solved using Gurobi, see Section 2-2-2. The outputs of the optimization model are x_{opt}, y_{opt}, J_{opt}. x_{opt} represents the optimized disposition timetable, y_{opt} contains all the Train Dispatcher (TD) decisions of the optimized system. J_{opt} is the optimized objective; the minimized total passenger delay.

code: *Appendix A-9 state to code which uses Gurobi to minimizes to objective with the selected constraints.*

Chapter 6

Case Study and Comparison

The previous chapter proposes a Delay Management (DM) model which reroutes passengers offline. This chapter tests this method and compares it to alternative methods. This answers the following sub-question:

Is the proposed approach an improvement on the current approaches?

The outline of this chapter is as follows: the first section gives an overview of alternative methods which will be used for comparing the proposed model of the previous chapter. Section 6-2 is on the test cases on which the different approaches will be tested with different delay scenarios. The third Section, 6-3 discusses different delay scenarios and the results of different approaches, followed by a section on the computer used for testing and the computation times. Finally, Section 6-5 summarizes the results and answers the corresponding research question.

6-1 Benchmark approaches

This section states the different approaches which will be used to compare the proposed solution. The constraints of these approaches will be the same, differentiation occurs in the objective[1]. There is a large number of dispatcher approaches that can be used for comparison [5]. The most common approaches are the no wait and the always wait approaches [19]. The basic approach from Section 2-3 will also be used as a benchmark objective. The online rerouting approach is not used as a benchmark, this is due to the unavailability of this code and because it is already proven to be too slow as stated in Section 2-4.

No Wait No Wait (NW) is the dispatcher's approach of deviating as little as possible from the scheduled timetable, only delaying scheduled events for feasibility reasons. The objective

[1]The transfer constraints are also included (with small weights) in the approaches which do not take them into account. Otherwise, the connection would be broken by default.

is formulated as follows:

$$J_{NW} = \sum_{i \in \mathcal{E}} x_i \tag{6-1}$$

Always wait The Always Wait (AW) approach focuses on one transfer affected by the delay and ensures the connection to always maintained. This is accomplished by changing the upper bound of the decision variable to 0. The objective of this approach is the same as the NW approach.

Basic Delay Management This approach is described in Section 2-3; where the assumed delay of a passenger missing their transfer is T_p, the periodicity of the connecting train.

6-2 Case study details

This section describes the details of the case study. First, the layout of the test cases is discussed, followed by the timetable and concluded with noteworthy details of the case study.

6-2-1 Layout

Figure 6-1 shows the part of the Dutch railway network which is selected as a case study for this delay management problem. This section is used because it provides a large number of transfers and possibilities of alternative routes as a result of the triangle form at Den Haag Centraal, Den Haag NOI and Den Haag HS. The used stations are chosen because they are used for changing tracks.

6-2-2 Timetable

The timetable information is from a manifest of the Dutch railways of 2011. One hour is extracted from this manifest, the resulting timetable is to be cyclic. The cyclic timetable consists of 11 different train lines which are split up into 160 train runs, see appendix B-1. The cyclic timetable can be summarized with Table 6-1 which shows the routes travel time and frequency of the trains lines.[2]

6-2-3 General case details

Besides the timetable, there are other general test details, these are stated in this subsection.

[2]This table is a simplification of the real-time, the actual time table in B-1 has within some train lines small deviations for feasibility reasons

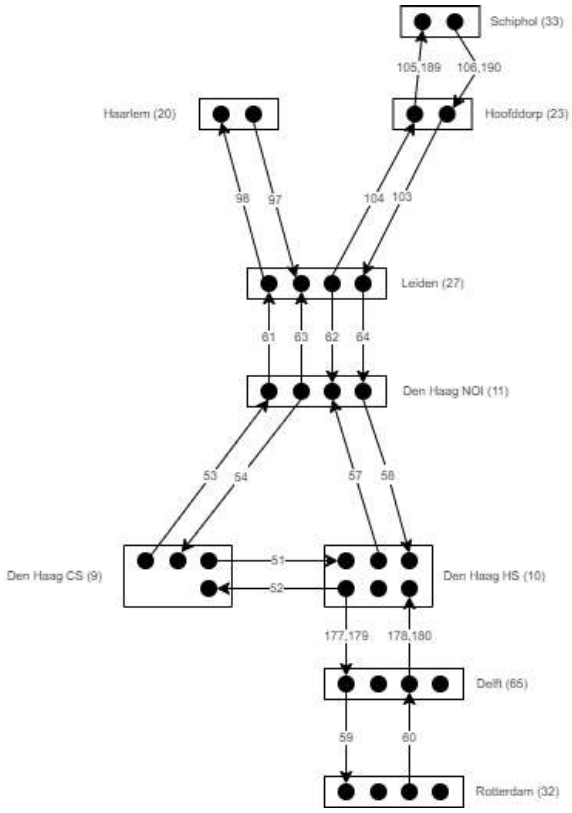


Figure 6-1: Case Study

Constraints details The maximum delay is assumed to be 30 minutes. The headway times and transfer times are assumed to be the same for all headways and transfers. The universal minimum headway time is 3 minutes between different trains on the same track. All train runs using the same track and with departure times of 30 minutes or less in between are constrained. The minimum universal transfer time between the arrival of the feeder train and the departure of the connecting train is 2 minutes.

As a result of some of the train lines being cut off, not all stations at the end of a train line are the actual turning points. The arrival and departure events at these stations are therefore not connected with dwell constraints.

Test details Applying the pathfinding algorithm from the previous chapter on the cyclic timetable produces 294 different OD pairs with departure times between 0 and 60 minutes. 120 of the 294 Origin-Destination (OD) pairs have a transfer.

Simulation For the optimization, a horizon of two hours is used. There are 331 train runs (662 events) and 136 unique transfers and 1444 headway decision variables in this interval. These are connected with 7286 constraints.

From	To	Line Numbers	Travel Time	Frequency [per hour]
Schiphol	Rotterdam	6	52	1
Schiphol	Rotterdam	92	48	1
Haarlem	Rotterdam	21,22	53	4
Den Haag CS	Rotterdam	19	23	2
Den Haag CS	Rotterdam	50,51	30	4
Den Haag CS	Schiphol	26,37	30	4
Den Haag CS	Schiphol	43	45	2
Den Haag CS	Haarlem	63	42	2

Table 6-1: Timetable information

6-3 Delay scenarios

This section discusses several delay scenarios used for testing the different DM strategies. For simplicity, the number of OD pairs with passenger weight attached is reduced to the minimum required for testing each scenario. This results in more understandable test results.
Each scenario has at least two weighed OD-pairs:

OD Pair 1: Path from an origin station o to a destination station d transferring at a station c.

OD Pair 2: Path from the transfer station c of OD pair 1 to the destination station d of OD pair 1.

The arrival of OD pair 1 at station c will be increasingly delayed, resulting in the wait-or-depart decision. This method will be used to find out at which magnitude of delay the connecting train no longer waits. The total passenger delay of the simplified scenarios can be formulated in the following equation:

$$J(x,y) = (x_d - \Pi_d) * (w_{p_2} + w_{p_1}(1 - y_{p_1})) + w_{p_1}\theta_{p_1}y_{p_1} \tag{6-2}$$

Where x & y are the sets of real-timetable and binary-decision variables respectively. x_d is the time of the planned arrival event OD-pair 1 and 2 at the destination station. w_{p_1} & w_{p_2} are the number of passenger traveling OD pair 1 and 2 respectively. y_{p_1} is the binary transfer variable of OD pair 1 which indicates whether the transfer is maintained ($y_{p_1} = 0$) or broken $y_{p_1} = 1$. θ_{p_1} is the actual delay a passenger of OD pair 1 experiences if a connection is broken. The offline rerouting and basic approach will both use this formula as objective, the difference between the two approaches is the value of θ_{p_1}[3]. All the other arrival and departure events and connection constraints will be weighed by a small number to ensure their minimization without them interfering with the test. The no wait and always wait do not take into account the passenger delay and will therefore be implemented, as stated in section 1-3.

6-3-1 Scenario 1: Alternative train line

Scenario 1 is to illustrate the delay of passengers is not always the frequency of the connecting train. In this scenario, OD-pair 1 departs at 7 from Rotterdam (32) to Den Haag CS (9) with

[3] as discussed in section 1-3

Relevence	id	Line	track	dep	arr	run	run slack	dwell	dwell slack	station 1	station 2
OD-pair 1.1	66	22	60	7	21	12	2	2	-	32	65
OD-pair 1.1	67	22	180	21	27	6	0	0	0	65	10
track 57	68	22	57	29	30	1	0	2	0	10	11
	142	51	60	2	17	14	1	2	-	32	65
	143	51	178	17	29	12	0	0	0	65	10
OD-pair 1.2.O	144	51	52	30	33	2	1	1	0	10	9
track 180	17	19	180	28	34	6	0	0	-	65	10
OD-pair 1.2.A	18	19	52	35	38	2	1	1	0	10	9
track 180	164	92	180	25	31	6	0	0	-	65	10
track 57	165	92	57	34	35	1	0	2	-	10	11

Table 6-2: Relevant train runs scenario 1

Id	Line	Start	Cost
136	51	0	3
24	19	5	3
130	50	15	3
144	51	30	3
18	19	35	3
124	50	45	3

Table 6-3: Shortest paths HS (10) to CS (9)

a transfer at Den Haag HS (10) weighted by 100 passengers. OD-pair 2 departs at 30 from Den Haag HS (10) to Den Haag CS (9) weighed by 50 passengers. All relevant[4] train runs are shown in table 6-2 the original connection is shown highlighted with green and red the alternative route in case green has a large delay is to transfer to line 19 (highlighted yellow). In this scenario, the train run 67 (id) will be delayed. The question becomes: "At what magnitude of delay will the different approaches break the connection?"

Alternative route & Objective

The offline approach computes the delay of missing the transfer to be 5 minutes; by taking the train which departs at 35 with a similar travel time. This is also shown in table 6-3 which shows the shortest path labels between Den Haag HS (10) and Den Haag CS (9). The above-mentioned information with equation 6-2 can be combined to get the following total passenger delay function:

$$J_{\text{scenario}_1}(y, x) = (x_d - 33) * (100 + 50 * (1 - y_{p_1})) + 50 * 5 * y_{p_1} \tag{6-3}$$

The basic approach will use as delay the frequency of the connecting train which would be 15 minutes, see table 6-3.

Result & comparison

Figure 6-2 shows the results of the different DM approaches in scenario 1. The no-wait benchmark test breaks the connection after an initial delay of 1 minute, this is because there was a dwell slack of 1 minute[5], see table 6-2. Applying this method for an initial delay greater than 1-minute results in a total passenger delay of $250 (= 50 * 5)$ minutes. The connecting train run 144 has 1 minute of run slack this results in the other method not accumulating passenger delay till the point of an initial delay of 2 minutes. The offline method breaks the connection after approximately 3.66 minutes of initial delay; at this point the 150 passengers in the connecting train line 51 experience a delay of 1.66 minutes which is approximately equal

[4] Train runs which influence delay management due to feasibility constraints. Also, the alternative route train runs.

[5] Taking into account the minimum required transfer time of 2 minute.

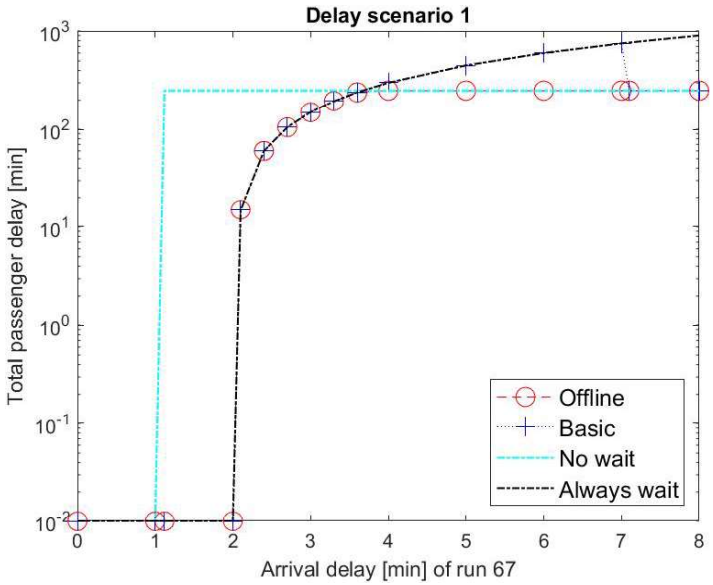


Figure 6-2: Results scenario 1 (alternative line)

to the delay caused by breaking the connection ($150*1.66 \approx 50*5$. The basic assumes a delay of 15 minutes for passengers of OD-pair 1 cause the connection is broken. Therefore, holding the connection till 7 minutes of initial delay; a $7-2=5$ minute arrival delay for passengers in train line 51 which is approximately equal to the assumed passenger delay of breaking the connection $150*5 \approx 50*5$. The highest total passenger delay of the online rerouting approach is 750 minutes whereas the highest total passenger delay of the offline rerouting approach is 250; worst case scenario an unnecessary passenger delay of 500 minutes.

6-3-2 Scenario 2: remain seated

The previous scenario shows that a train line of a different frequency can provide an alternative route when a connection is broken. Scenario 2 illustrates the alternative route of remain seated, see Section 4-3.

For this example OD-pair 1 is from Schiphol (33) to Rotterdam (32) transferring at Den Haag HS (10); transferring from line 92 to line 19. OD-pair 2 is from Den Haag HS (10) to Rotterdam (32) using line 19. OD pairs 1 and 2 are weighed with 100 and 50 people respectively[6]. All relevant train run information of this scenario is shown in table 6-4. Lines 22 and 51 are also included because of the feasibility constraints. Table 6-4 shows that the transfer from train run 160 (green) to 14 (red) has 0 transfer slack; 2 minutes is the minimum required transfer time. The table also shows that line 19 has one minute of run slack between station Den Haag HS (10) and Rotterdam (32). This lack of slack is one reason this scenario is chosen.

[6]Similar to the previous scenario

id	type	track	dep	arr	run	run slack	dwell	dwell slack	station 1	station 2
13	19	51	83	85	2	0	10	-	9	10
14	19	177	87	93	6	0	2	0	10	65
15	19	59	93	106	12	1	0	0	65	32
160	92	58	82	85	2	1	0	-	11	10
161	92	179	87	96	4	5	2	0	10	65
162	92	59	96	109	11	2	0	0	65	32
61	22	58	86	90	1	3	1	-	11	10
62	22	177	93	99	6	0	2	1	10	65
63	22	59	99	112	12	1	0	0	65	32
137	51	51	88	92	2	2	10	-	9	10
138	51	179	93	102	9	0	1	0	10	65
139	51	59	102	118	14	2	0	0	65	32

Table 6-4: Relevant train runs scenario 1

id	line	start	cost
50	22	3	19
40	21	18	20
14	19	27	19
62	22	33	19
29	21	48	19
20	19	57	19

Table 6-5: Shortest paths HS (10), Rotterdam (32)

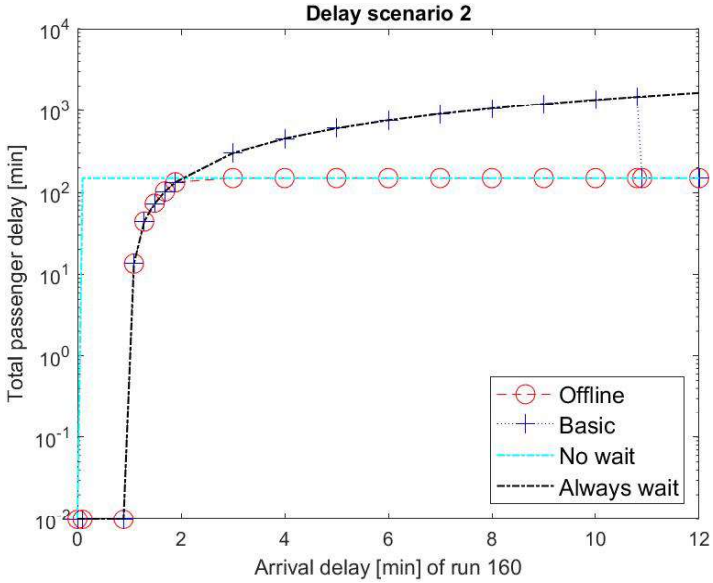


Figure 6-3: Results scenario 2 (remain seated)

Alternative route & Objective

The alternative route computed with the offline approach is to remain seated in train line 92 arriving at Rotterdam (32) at 109 instead of the planned 106; a 3-minute delay see table 6-4. The offline method takes the periodicity of the connecting train line 19 as a delay; 30 minutes, see 6-5. The alternative of the offline approach is not visible in the shortest path output but is generated with the alternative path calculation, see section 4-3. Above information can be used to fill in equation 6-2 resulting in the following total passenger delay function:

$$J_{\text{scenario}_2}(y, x) = (x_d - 106) * (100 + 50 * (1 - y_{p_1})) + 50 * 3 * y_{p_1} \tag{6-4}$$

relevance	id	line	track	dep	arr	run	run slack	dwell	dwell slack	station 1	station 2
	47	22	97	29	46	17	0	2	-	20	27
OD pair 1.1.O	48	22	62	49	55	6	0	2	1	27	11
OD pair 1.1.A	49	22	58	56	60	1	3	1	0	11	10
	152	63	64	47	59	11	1	2	-	27	11
OD pair 1.2.O	153	63	54	60	64	2	2	1	0	11	9
OD pair 1.2.A	24	19	52	5	8	2	1	1		10	9
	73	26	64	55	63	5	3	1	-	27	11
	74	26	54	3	7	2	2	0	0	11	9

Table 6-6: Relevant train runs scenario 2

Result & comparison

Figure 6-3 shows the results of the different approaches for different delays. As mentioned previously there is 0 dwell slack, this results in the no-wait benchmark approach breaking the connection after just a small initial delay. The run slack is 1 minute therefore passenger delay occurs after 1 minute of the initial delay. The offline approach breaks the connection after 2 minutes of the initial delay. The 150 passengers of line 19 would experience 1 minute of delay[7].: this total passenger delay of $150(= 150*1)$ equals the cost of breaking the connection $150(= 50 * 3)$.

The basic approach assumes the cost of breaking the connection to be $1500(= 50 * 30)$; 10 times larger than that of the offline approach. Figure 6-3 shows that the basic approach holds the connection till an initial delay of approximately 11 minutes. Where the total passenger delay is around 1500 minutes; 10 times larger than that of the offline approach.

6-3-3 Scenario 3: remain seated then transfer

Scenario 2 shows an instance when the best alternative route for passengers is to just remain seated in the feeder train. In this scenario, the best alternative path is to remain seated in the feeder train then transfer to a later station, see Section 4-3

This scenario has OD-pair 1 passengers travelling from Haarlem (20) to Den Haag CS (9) transferring from train line 22 to 63 at Leiden (11). OD-pair 2 is from Leiden (11) to Den Haag HS (32). Similar to the other scenarios is the number of passengers of OD-pairs 1 and 2, 100 and 50 respectively. The delay in this scenario will be introduced at the arrival of train line 22 at Leiden (11).

Alternative route & Objective

The alternative route for OD-pair 1 in case the connection is broken is remaining seated for one station to Den Haag HS (10) there transferring to line 19 to Den Haag CS (9). The arrival will be 68 instead of 64; a 4-minute delay. This in combination with equation 6-2 results in the following simplified function for total passenger delay:

$$J_{\text{scenario}_2}(y, x) = (x_d - 64) * (100 + 50 * (1 - y_{p_1})) + 50 * 4 * y_{p_1} \tag{6-5}$$

[7] 2 minutes of initial delay minus 1 minute of run slack.

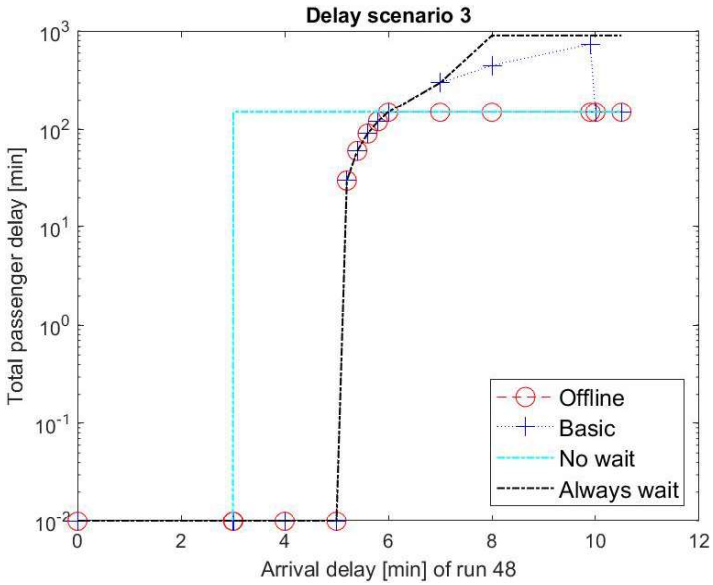


Figure 6-4: Results scenario 2 (remain seated than transfer)

The alternative to offline alternative uses the delay of 15 minutes; the periodicity of line 63.

Result & comparison

Table 6-6 show there is a run slack of $3(= 60-55-2)$ minutes. The no-wait approach will not wait and directly result in a passenger delay of $200(= 50*4)$ minutes. There are 2 minutes of run slack in train run 153; therefore passenger delay at the other approaches occurs after the $5(= 3+2)$ minutes of the initial delay. After the 5 minute mark, the connection is held by the offline approach for approximately 1.33 minutes, at that point, the cost of breaking the connection becomes equal to the delay of passengers in the feeder train; $50*4 \approx 1.33*150$. The basic approach breaks the connection around the 10 minutes initial delay point, at this point, the delay of passengers in the connecting train would be equal to the predicted cost of breaking the connection; $750(15*50)$. A strange phenomenon occurs with the always-wait approach, by ensuring the forced wait of the connecting train results in the order of train run 153 and 74 highlighted in red to switch after an initial delay of 8 minutes. This is the result of this benchmarking approach not taking into account the number of passengers on the train.

6-4 Test setup and computation time

All simulations are solved with Gurobi 9.1.2 on a laptop with an Intel(R) Core(TM) processor of 2.2 GHz and a RAM of 16 GB. The computation time of one delay scenario in all cases was of less than 0.1 second.

6-5 Conclusion

The goal of this Chapter 6 was to test the proposed objective, the offline approach, to answer the research sub-question:

Is the proposed objective an improvement on the current approaches?

The current approaches are the basic approach and the benchmark approaches described in section 2-3. This section also mentions the online rerouting approach which is not used as a benchmark for testing because of the unavailability of the code. The time it would take to implement and the certain longer computation time as stated in section 2-4.

A section of the complete Dutch railway system is used as a test case, see section 6-2. Three scenarios are used for testing and comparing the performance of the offline approach against the benchmark approaches. The results and comparison of each scenario state that the offline approach outperforms the benchmark approaches in regards to the total passenger delay.

The following conclusions can be drawn from the results & comparisons of the different scenarios. The no-wait approach is the best for a robust network but results in a lot of passengers unnecessary missing their connection. Whereas the always-wait approach can hold the connection till the point the network becomes unfeasible. The basic approach takes into account the possibility of holding or breaking a connection. However, this method uses a wrongfully high cost for breaking a connection, due to the assumption that the delay is the periodicity of the connecting train while the actual delay can be smaller. This higher cost for breaking a connection results in transfers being held longer than needed.

The more accurate delay assumption of the offline approach results in minimal passenger delay, thereby proving that the offline approach is an improvement on the basic approach. Thereby answering the research sub-question.

A possible improvement on these tests will be a time domain in which the alternative route is feasible. For example, in the first scenario, the alternative scenario is feasible until an initial delay of 5 minutes. After 5 minutes the transfer time between line 22 and line 19 becomes too small. Resulting in either the need for line 19 to wait for the people transferring from line 22. The other option is that passengers should transfer to train line 50 departing at 45. This is currently not in the first-order approximation, this is left to further research.

Chapter 7

Concluding remarks

At last, the final chapter of this thesis reviews all the previous chapters and gives some concluding remarks. This chapter starts with the conclusion, followed by the contributions of this work and concludes with a the possibilities for future work.

7-1 Conclusions

The thesis aimed to answer the following question:

> Is it possible to develop a delay management method which accurately reroutes passengers and still computes in a reasonable time?

This section first answers the sub research questions and answers thereafter the above stated main research question.

7-1-1 What is the currently available literature lacking?

Chapter 2 reviews the current state of railway delay management, to answer the relevance of the thesis. This chapter concludes that there is a gap in the literature between a slow accurate and a fast inaccurate approach. The basic model lacks accuracy as a result of the basic approach for rerouting passengers. Whereas the online approach has a computation time which exceeds the time of a Train Dispatcher (TD) to make a decision. This is the result of a complex model that reroutes the passengers simultaneous with the delay management by adding a lot of variables and constraints. Chapter 2 is concluded with a theoretical approach that reroutes the passengers offline which could fill this gap.

7-1-2 How to route and reroute passengers through a network?

Chapter 3 states the theory of time-dependant rerouting. Chapter 4 proposes an algorithm that computes the shortest paths in the network and also provides an alternative path for those paths which contain a connection. This chapter introduces two alternatives paths which can be applied to give a more accurate view of the delay of passenger who miss their connection.

7-1-3 How to combine passenger rerouting and delay management?

Chapter 5 combines the theory of Chapter 2 and outputs of the proposed (re)routing algorithm of Chapter 4 to a Mixed Integer Linear Programming (MILP) which can be solved using Gurobi. This chapter also refers to the code used, stated in the appendix.

7-1-4 Main research question

Is it possible to develop a delay management method which
accurately reroutes passengers and still computes in a reasonable time?

Chapter 6 shows the proposed method outperforms the benchmark methods as a result of the more accurate and lower cost of breaking a connection. Chapter 5 also stated that the computation time of the proposed method will be in the same order as the computation time of the basic approach. The proposed research thereby fulfils all the requirements of the computation time while also outperforming the basic approach in reducing the total passenger delay.

7-2 Contribution

The main contribution of this thesis is a proposed pathfinding algorithm for the routing and rerouting of passengers in a time-dependent network and a delay management model which is generated with the output of the pathfinding algorithm. The proposed delay management model is an improvement on the current state-of-the-art since it accurately reroutes passengers while still computing in a reasonable time.

This thesis also provides the code for generating constraints from timetable information. Alternative test case can easily be tested using this code. This is convenient for further research.

7-3 Future work

Even though the proposed algorithm shows great results for passenger-focused delay management of railway networks, several challenges and opportunities for future research remain. This section outlines some recommendations for future work.

Second order approach

As stated in 6-5 the alternative paths are feasible within a certain magnitude of delay. With large or multiple delays this approach might wrongfully use a low cost for breaking a connection while in these cases the actual delay is larger than calculated prior. Possible solutions might be a more complex objective or more constraints. Another approach is to verify feasibility after computation and reruns the optimization with an updated objective. This is a valid option since the computation time of the proposed optimization method is low, an updated objective will have a smaller effect on computation time than the first option of using a more complex objective or additional constraints. Speedup techniques can be applied to reroute faster [2].

frequency labels

The current approach only assumes a common periodicity of T_{Π}. The theory of frequency labels can reduce the overall number of labels by merging labels which have a have a periodicity of a fraction of the original T_{Π}. More complex test cases where frequency only holds in a given time interval can also be modelled using the frequency labels, see Section 3-2-4.

Distributed

Distrubted optimazation is a solution if, in future work, computation time of the MILP increases as a result of the increased complexity of the objective or as an result of larger test cases. [18].

User Interface for implementation as Decision Support System

As mentioned in 1-3 the purpose of the research is to be implemented as a Decision Support System (DSS). This can be accomplished by creating a friendly user interface (UI) in which the railway network, trains, passenger streams, ordering of trains and connections are visually represented.

Real-time network data

The NS has an API interface from which live data can be extracted. The real-time data would further prove the potential of this approach and might result in a joint project between the TU Delft and ProRail.

Add complexity

The proposed approach already implements the headway constraint which is not common in the field of delay management. Some of the following alternative complexities can also be added to the proposed method:

1. Rescheduling of rolling stock; platform allocation

2. Taking crew scheduling into account
3. Adding additional weight multipliers on delay resulting in waiting on a platform vs waiting in a train.
4. adding train capacity (and spill of passengers) [20]

Appendix A

Code

This appendix contains all the codes used in this thesis. Most of the code is also referred to in the different chapters.

A-1 Load data

The timetable of periodicity T_{Π} is assumed to be delivered in excel from with With two sheets; timedata and station. The timedata sheet should contain the following:

- **ID:** train run id.
- **Type:** train runs are grouped by type/line.
- **Train:** a train type can have divisor of T_{Π} as periodicity. This will result in the train type repeating itself to fill the period $[0, T_{\Pi}]$. When a test case has a train type which is cut off (the begin or end point is not the a turning point) a new train id is introduced to form the the dwell constraints properly, see 2-1-2.
- **Track:** track id.
- **Departure and arrival station:** station id of the departure and the arrival station.
- **Departure and arrival time:** the scheduled departure and arrival time.
- **Run time:** minimum time between the departure and the arrival time (for rescheduling), see 2-1-2.
- **Dwell time:** minimum dwell time between the arrival time of the previous train run of the same type and the departure time of the current train run, see 2-1-2.
- **Shift:** all the departure times have to be in the interval of $[0, T_{\Pi}]$. If the actual departure time is be higher than T_{Π} a shift of $-T_{\Pi}$ is used to ensure the previous rule, shift is in this case set to 1. Otherwise shift is 0.

The station sheet couples station id to the corresponding station name. An example of the timedata sheet is shown in B-1.

Code: load_data.m

```
%% read excel
model='model07_4_3.xls';
timedata            = sortrows(readmatrix(model,'sheet','timedata'));
station.name        = readcell(model,'sheet','stations');
% timedata(end,:)=[];  %remove last line of table if empty

%% define collums
info.id          = 1;
info.type        = 3;
info.train       = 18;
info.track       = 4;
info.depart      = 7;
info.arrive      = info.depart+1;
info.run         = 9;
info.shift       = 10;
info.dwell       = 11;
info.origin      = 13;
info.destination= info.origin+1;

%% create station and type structers
station.list=unique(timedata(:,info.origin));
station.length=length(station.name);
type.frequency=[];
type.list=unique(timedata(:,info.type));
type.max=max(type.list);
type.length=length(type.list);
type.frequency(type.list,1)=60;
```

A-2 Preprocess timedata

The preprocessing is consist of; forming a time dependant graph, combine train runs to a train line and to ensure that the output is First In First Out (FIFO), see Section 4-1-2. The time dependant graph and the connecting of the train lines is done by the code "make_time_dependant_graph.m". The graph is made FIFO with the code "make_fifo.m". The resulting graph is represented by cell cost.e; the cell data type is used because the number of labels attached to an edge can vary.

Code: make_time_dependant_graph.m

```
function [segment,station,type,cost,extra] = make_time_dependant_graph(
    timedata,station,type,info)
%MAKE_TIME_DEPENDANT_GRAPH transforms timetable data into a time
    dependant
%graph
```

```
4  %   This function also combines trainruns into train lines (type=line)
5  %   info.line is not used.
6  %   Required: trainruns are grouped by type,
7  %   label = [id,start, frq, cost, #connections,previous_id]
8  %   Output cost.e=graph with multiple labels on the edges
9  %% deffine varables
10 cost.e          = cell(station.length);   %cost labels
11 cost.e_max      = inf(station.length);    %maximum time to travel
12 cost.e_max(1:1+station.length:end)= NaN;    %make diag NaN
13 cost.e_min      = NaN(station.length);    %minimum travel time
14 cost.e_count    =zeros(station.length);
15 cost.transfer     = 2; %2 min overstapp
16 types           = unique(timedata(:,info.type))';       %extract
       line_numbers
17 segment.length  = length(unique(timedata(:,info.id)));  % amount of train
       runs
18 segment.line    = zeros(1,segment.length);
19 segment.o_station = zeros(1,segment.length);
20 station.lines   = cell(1,station.length);
21 type.stations   = cell(1,type.max);
22 type.segments   = cell(1,type.max);
23 extra.not_added = [];
24 extra.removed   = [];
25 %% iterate trough segments
26 for type_number = types
27     type_temp           = timedata(timedata(:,info.type) == type_number
           ,:);
28     frequency           = type.frequency(type_number);
29     length_type_i       = size(type_temp,1);
30     count=0;
31     for i = 1:length_type_i
32         %% add label (track)
33         track       = type_temp(i,:);
34         run         = track(info.id);
35         origin      = track(info.origin);
36         destination = track(info.destination);
37         arrive      = track(info.arrive);
38         [id,cost]   = id_labels(origin,destination,cost);
39         start       = track(info.depart);
40         start_rel   = mod(start,frequency);
41         track_cost  = arrive-start;
42         cost_temp   = [id,run,start_rel,frequency,track_cost,0,0];
43         [~,cost,extra] = ...
44             add_edge1(origin,destination,cost_temp,cost,extra);
45         segment.line(run)=type_number;
46         segment.o_station(run) = origin;
47         segment.d_station(run) = destination;
48         %% train line
49         if i==1 && track(info.dwell)==0
50             error('start of type has dwell of 0')
51         end
52         if i>1
53             if type_temp(i-1,info.origin)==type_temp(i,info.destination)
```

```
54                  count=0;
55              end
56          end
57          if count>0
58              shift=track(info.shift);
59              for j =1:count
60                  track_0     = type_temp(i-j,:);
61                  run_0       = track_0(info.id);
62                  origin_0    = track_0(info.origin);
63                  start       = track_0(info.depart);
64                  start_rel   = mod(start,frequency);
65                  track_cost  = arrive-start-60*shift;
66                  [id,cost]   = id_labels(origin_0,destination,cost);
67                  cost_temp   = [id,run_0,start_rel,frequency,track_cost
                        ,0,0];
68                  [~,cost,extra] = ...
69                       add_edge1(origin_0,destination,cost_temp,cost,extra);
70                  shift=shift+track_0(info.shift);
71              end
72          end
73          count=count+1;
74          %% add orgin station to line.station if not already incuded
75          if ~any(type.stations{type_number}==origin)
76              type.stations{type_number}(end+1) = origin;
77              station.lines{origin}(end+1) = type_number;
78          end
79          %% add segment to line
80          type.segments{type_number}(end+1)= run;
81      end
82      %check if lines start at turningpoint
83      if ~(origin == type.stations{type_number})
84          error("Line did not start at the turning point.")
85      end
86  end
87
88  %% check if lines are not tracks are not added
89  if ~isempty(extra.not_added)
90      error("A track has not been added.")
91  elseif ~isempty(extra.removed)
92      error("A track has been removed.")
93  end
94  end
95  function [id,cost] = id_labels(O,D,cost)
96  % makes id's for within one OD pair
97  cost.e_count(O,D) =cost.e_count(O,D) +1;
98  id = cost.e_count(O,D);
99  end
100 function [add_label,cost,extra] = add_edge(O,D,label,cost,extra)
101 if label(5) >= cost.e_max(O,D)
102     %checks if not adding a longer route
103     error("track takes longer then existing max")
104 end
105 % add min and max
```

```
106 cost.e_min(O,D)    = min(cost.e_min(O,D), label(5));
107 cost.e_max(O,D)    = min(cost.e_max(O,D), sum(label(4:5)));
108 add_label          = true;
109 % remove            = false;
110 cost.e{O,D} = [cost.e{O,D};label];
111 end
```

Code: make_fifo.m

```
1  %% Make the time dependt graph fifo by removing non fifo labels
2  removed=[];
3  removed_list=[];
4  for i=1:station.length
5      for j=1:station.length
6          if ~isempty(cost.e{i,j})
7              %% Remove redundant lines
8              cost.e{i,j}=sortrows(cost.e{i,j},[3 5]);
9              remove=[];
10             labels=cost.e{i,j};
11             for k=2:size(labels,1)
12                 if labels(k,3)==labels(k-1,3)
13                     % same start remove longer travel
14                     remove(end+1)=k;
15                 elseif sum(labels(k,[3 5]))<=sum(labels(k-1,[3 5]))
16                     % remove earlier start later arrival
17                     remove(end+1)=k-1;
18                 end
19             end
20             %k = end
21             if sum(labels(1,[3 5]))+60<=sum(labels(k,[3 5]))
22                 remove(end+1)=k;
23             end
24             if ~isempty(remove)
25                 removed_list=[removed_list;i*ones(length(remove),1),j*
                       ones(length(remove),1), ...
26                     cost.e{i,j}(remove,:)];
27             end
28             cost.e{i,j}(remove,:)=[];
29             removed=[removed,remove];
30
31             %% acurate max
32             start=cost.e{i,j}(:,3)+1;
33             wait_afterer_departure= min(weigth_of_labels(start,cost.e{i,j
                   }))+1;
34             cost.e_max(i,j)=max(wait_afterer_departure);
35         end
36     end
37 end
```

A-3 Shortest path algorithm

The shortest paths are computed with the following algorithm "shortest_paths.m". In this function the code "run_Q_top.m"is used to evaluate the top row off Q, see Section 4-2. To find the new maximum the different labels are evaluated using the function "weigth_of_labels.m", see Appendix A-5. The output of the shortest path algorithm is graph cost.n with the frequency labels for the shortest paths.

Code: shortest_paths.m

```
function [cost,segment] = shorthest_paths(cost,segment,station)
%SHORTHEST_PATHS compete all shortest path in netwerok
% departure time are in [0,T_\Pi]
cost.n          = cell(station.length);
cost.n_max      = inf(station.length);    %maximum time to travel
cost.n_max(1:1+station.length:end)= NaN;  %make diag NaN
cost.n_min      = NaN(station.length);    %minimum travel time
Q=[];
cost.n=cost.e;
cost.n_max=cost.e_max;
cost.n_min=cost.e_min;
%% run one-to-all shortest path algorithm multiple times
for origin = 1:station.length
    %% initialazi existing paths from the origin
    for i=find(cost.e_min(origin,:)>0)
            cost.Q{origin,i} = cost.e{origin,i};
            Q     = [Q ; cost.e_min(origin,i),0,i];
    end
    Q = sortrows(Q) ;    %order Q on first row; dist to origin
    while ~isempty(Q)
        [Q,cost] = run_Q_top(origin,Q,cost);
    end
end
end
```

Code: run_Q_top.m

```
function [Q,cost] = run_Q_top(O,Q,cost)
%RUN_Q_TOP Evalute the top of the queue (Q)
% If new shortest paths update Q
% Finallty remove used top row of Q
% label = [id,track,start, frq, cost, #connections, previos station]
C       = Q(1,3);
labels_OC  = cost.n{O,C};
for D=find(cost.e_min(C,:)>0)
    labels_CD  = cost.e{C,D};
    added=false;
    transfers=0;
    temp_min=inf;
    if C~=D && cost.n_min(O,C)+cost.e_min(C,D)<cost.n_max(O,D)
```

```
14          start           = labels_OC(:,3);
15          start_C         = start+labels_OC(:,5)+cost.transfer;
16          cost_CD         = min(weigth_of_labels(start_C,labels_CD));
17          new_cost_OD     = labels_OC(:,5)+cost.transfer+cost_CD';
18          if isempty(cost.n{O,D})
19              added=true;
20              transfers=1;
21              new_labels      = labels_OC;
22              new_labels(:,5) = new_cost_OD;
23              new_labels(:,6) = 1;
24              new_labels(:,7) = C;
25              cost.n{O,D}     = new_labels;
26              temp_min        = min(new_labels(:,6));
27          else
28              %cost is not empty
29              old_cost_OD     = min(weigth_of_labels(start,cost.n{O,D}))';
30              new_labels      = [];
31              index_shorter   = find(new_cost_OD<old_cost_OD);
32              if ~isempty(index_shorter)
33                  added=true;
34                  for i = index_shorter'
35                      new_label  = labels_OC(i,:);
36                      new_label(5) = new_cost_OD(i);
37                      new_label(6) = new_label(6)+1;
38                      new_label(7) = C;
39                      new_labels = [new_labels;new_label];
40                      transfers  = min(new_label(6),transfers);
41                      temp_min   = min(new_label(5),temp_min);
42                  end
43                  cost.n{O,D} = [cost.n{O,D}; new_labels];
44              end
45          end
46      end
47      %% if added: make labels fifo, make new min and max(re)instate D into
            Q
48
49      if added
50          [cost]              = make_fifo_labels(O,C,D,cost);
51          %min & max
52          cost.n_min(O,D)     = min(cost.n{O,D}(:,5));
53          [cost]              = max_path(cost,O,D);
54          %(re)instate D to Q
55          if ~any(Q(:,3)==D)
56              Q=[Q;temp_min,transfers,D];
57          else
58              index_Q = find(Q(:,3)==D);
59              if temp_min < Q(index_Q,1)
60                  Q(index_Q,:)=[];
61                  Q=[Q;temp_min,transfers,D];
62              end
63          end
64          Q = sortrows(Q) ;
65      end
```

```
66      %%
67
68
69  end
70  Q(1,:) = [];
71  end
72
73  function [cost]=make_fifo_labels(O,D,cost)
74  cost.n{O,D}=sortrows(cost.n{O,D},[3 5]);
75  running=true;
76  while running
77      remove=[];
78      labels=cost.n{O,D};
79      changed=false;
80      for k=2:size(labels,1)
81          if labels(k,3)==labels(k-1,3) && labels(k,6)>=labels(k-1,6)
82              % same start remove longer travel
83              remove(end+1)=k;
84              changed=true;
85          elseif sum(labels(k,[3 5]))<=sum(labels(k-1,[3 5])) ...
86                  && labels(k,6)<=labels(k-1,6)
87              % remove earlier start later arrival
88              remove(end+1)=k-1;
89              changed=true;
90          end
91      end
92      if sum(labels(1,[3 5]))+60<=sum(labels(k,[3 5])) ...
93              && labels(1,6)<=labels(k,6)
94          %k is end of labels
95          remove(end+1)=k;
96          changed=true;
97      end
98      cost.n{O,D}(remove,:)=[];
99      running=changed;
100 end
101 end
102 function [cost] = max_path(cost,O,D)
103 %UNTITLED3 Summary of this function goes here
104 %   Detailed explanation goes here
105 start=cost.n{O,D}(:,3)+1;
106 wait_afterer_departure= min(weigth_of_labels(start,cost.n{O,D}))+1;
107 cost.n_max(O,D)=max(wait_afterer_departure);
108 end
```

A-4 Alternative paths

The alternative paths can be computed with the output of the shortest path algorithm. Each label with a connection is evaluated in the alternative path is compute with "alternative_path.m". Delay 1 is calculated using an alternative shortest path at the transfer station. delay 2 is calculated with the alternative path of remaining seated for an extra stop. See Section 4-3 for a larger elaboration.

Code: alternative_path.m

```
%labels with transfer
%label = [id,id_run,start, frq, cost, #connections,previous_id]
transfer_labels=[];
for O=1:station.length
    for D=1:station.length
        if ~isempty(cost.n{O,D})
            for i =1:size(cost.n{O,D},1)
                label=cost.n{O,D}(i,:);
                if label(6)>0
                    C=label(7);
                    if label(6)==1
                        %% delay 1 alternative same station
                        start_O=label(3);
                        freq= label(4);
                        cost_OC = min(weigth_of_labels(start_O,cost.n{O,C
                            }));
                        start_C = mod(start_O + cost_OC + cost.transfer,
                            freq);
                        index_CD= find(cost.n{C,D}(:,3) >= mod(start_C,
                            freq),1,'first');
                        if isempty(index_CD)
                            index_CD=1;
                            transfer_time = cost.n{C,D}(index_CD,3)-
                                start_C+60;
                        else
                            transfer_time = cost.n{C,D}(index_CD,3)-
                                start_C;
                        end
                        cost_CD2    = min(weigth_of_labels(start_C+
                            transfer_time+1,cost.n{C,D}))+1;
                        delay       = cost_CD2-cost.n{C,D}(index_CD,5);

                        %% delay2 alternative remain seated
                        type_temp=segment.line(label(2));
                        labels_temp=[];
                        for j=station.list'
                            if j~=O
                                labels_temp=[labels_temp;cost.n{O,j},j*
                                    ones(size(cost.n{O,j},1),1)];
                            end
                        end
```

```
35                          index_remove = find(labels_temp(:,2)~=label(2));
36                          labels_temp(index_remove,:)=[];
37                          index_remove=find(labels_temp(:,6)>0);
38                          labels_temp(index_remove,:)=[];
39                          labels_temp=sortrows(labels_temp,5);
40                          x=find(labels_temp(:,end)==C);
41                          if x==size(labels_temp,1)
42                              delay2=NaN;
43                          else
44                              C3=labels_temp(x+1,end);
45                              cost_OC3 = labels_temp(x+1,5);
46                              if C3==D
47                                  cost_CD3=0;
48                              elseif any(labels_temp(:,end)==D)
49                                  index_1=find(labels_temp(:,end)==D);
50                                  cost_CD3=labels_temp(index_1,5)-cost_OC3;
51                              else
52                                  start_C3=start_O+cost_OC3+cost.transfer;
53                                  cost_CD3 = min(weigth_of_labels(start_C3,
                                        cost.n{C3,D}))...
54                                      +cost.transfer;
55                              end
56                              cost_OD3 = cost_OC3+cost_CD3;
57                              delay2   = cost_OD3-label(5);
58                              if delay2>delay
59                                  delay2=NaN;
60                              end
61                          end
62                          transfer_labels(end+1,:)=[O,D,C,label(6)-1,
                                transfer_time,start_O,start_C+transfer_time,
                                delay, delay2];
63                      else
64                          transfer_labels(end+1,:)=[O,D,C,label(6)-1,NaN,
                                start_O,start_C+transfer_time,NaN,NaN];
65                      end
66                  end
67              end
68          end
69      end
70  end
71  transfer_labels=sortrows(transfer_labels,[5])
```

A-5 Evaluate labels

Evaluating labels is done by the function "weight_of_labels.m", using the evaluation method of Section3-2-4.

Code: weight_of_labels.m

```
function [t,labels] = weigth_of_labels(t,labels)
%WEIGTH_OF_LABELS evalute the weigth of group of labels
% label = [id,line, start,frq ,interval, cost]
% output is a matrix if mutple times and multible labels are as input
start           = labels(:,3);
p               = labels(:,4);
c               = labels(:,5);
weigth          = zeros(size(labels,1),length(t));
for i = 1:length(t)
    weigth(:,i) = start+ceil((t(i)-start)./p).*p-t(i)+c;
end
end
```

A-6 Make dayplan

A periodic timetable can be repeated to form a timetable this is done with the function "make_dayplan.m". A dayplan is generated from for the interval $[0, Nsim]$, $Nsim$ is for example the whole day; 24 hours.

Code: make_dayplan.m

```
function [dayplan,info] = make_dayplan(timetable,Nsim,info,type)
%MAKE_DAYPLAN Summary of this function goes here
% Uses timetable to make dayplan with the length of simulation time.
%   Timetable matrix counting different trainline information
%   Nsim      simulation time
%   T         periodicity of the timtable
% order =[info.line, info.type, info.track, info.dir1, info.dir2, ...
%     info.depart, info.arrive, info.run, info.shift, info.dwell, ...
%     info.joined, info.origin, info.destination]
Number_Of_TrainLines    = size(type.frequency,1);
dayplan                 = [];
train                   = 1;
track_time              = [info.depart info.arrive];
T       = 60;               %type.frequency(type_i);
train_ids=unique(timetable(:,info.train))';
for type_i = train_ids
    % trainruns are grouped by train ID
    runs    = find(timetable(:,info.train)==type_i);
    timedata_temp = timetable(runs,:);
    if isempty(runs)
        error("could not find lines %i",runs)
```

```
22      end
23      shift_begin_end = timedata_temp(end,info.arrive)+timedata_temp(1,info
            .dwell)>= ...
24          timedata_temp(1,info.depart);
25      som             = sum(timedata_temp(2:end,info.shift)==-1)+
            shift_begin_end;%shift sum
26      shift_temp      = [0;cumsum(-timedata_temp(2:end,info.shift))];
27      timedata_temp(:,track_time)=timedata_temp(:,track_time)+shift_temp*T;
28      cycles  = ceil(Nsim/T);
29      dayplan_temp=[];
30      if ~shift_begin_end
31          % if dwell puls arrival of the end of teh train line is grater
                than
32          % the deparuutre time of the beginning of the line shift with -1
33          dayplan_new         = timedata_temp;
34          dayplan_new(:,track_time)  = dayplan_new(:,track_time)+T*-som;
35          dayplan_new         = [dayplan_new train*ones(length(runs),1)];
36          dayplan_temp=[dayplan_temp;dayplan_new]
37          train=train+1;
38      end
39      for i = 1:cycles
40          %repeat train line multiple times
41              dayplan_new         = timedata_temp;
42              dayplan_new(:,track_time)  = dayplan_new(:,track_time)+T*(i
                    -1)*som;
43              dayplan_new         = [dayplan_new train*ones(length(runs),1)
                    ];
44              dayplan_temp            = [dayplan_temp;dayplan_new];
45              train               = train+1;
46      end
47
48      number_trains=dayplan_temp(end,end)-dayplan_temp(1,end)+1
49      dayplan_temp2=dayplan_temp;
50      if som>1
51          for j=2:som
52              dayplan_temp(:,end)=dayplan_temp(:,end)+number_trains;
53              dayplan_temp(:,track_time)=dayplan_temp(:,track_time)-60
54              dayplan_temp2=[dayplan_temp2;dayplan_temp];
55              train=train+number_trains;
56          end
57      end
58      dayplan=[dayplan;dayplan_temp2];
59  end
60  %% select departure times between 0 and Nsim
61  info.train = size(dayplan,2);
62  indT=find(dayplan(:,info.depart)>-0.1&dayplan(:,info.depart)<Nsim+0.1);
63  dayplan=dayplan(indT,:);
64  %% error massage if empty collums
65  if ~isempty(find(isnan(dayplan(1,:))))
66      msgbox("There are empty colloms in table delete empty collumns in
            excel if columns aftere empty collumns are used")
67  end
68  end
```

A-7 Dayplan 2 constraints

The code "dayplan2milp.m" make the timetable into constraints of the form of Section 2-2-2. The constraints are put in the form of $Ax \leq b$. The constraints have 5 different types: timetable, run, dwell, headway and transfer. These are number 1 to 5 respectively. The constraints are formed using the theory from Section 2-2-3.

Code: dayplan2milp.m

```
1  function [Avec,Bvec0,Bvec1,Rvec,Cvec,pvec,pname,pnum,A,b] = dayplan2milp(
       dayplan,beta,Nh,info,segment,type,cost)
2  %DAYPLAN2MILP transforms the timetable into the constraint of the MILP
3  %% Ax<=b
4  % Avec is for making sparse matrix, first row is constraint, second row
       variable
5  %   the third is the value and the fourth is type of constraint.
6  %   the type of constriants are numberd 1 tot 5
7  %   following the order timetable,run,dwell,headway and transfer
       constraint
8  %   b=Bvec0+pvec(Bvec1);
9  tau_headway=3;
10 n=size(dayplan,1);
11 nlines=dayplan(end,end);
12 i=1;
13 tela=1;
14 telb=1;
15 item=1;
16
17 %
       ================================================================================

18 %
19 % add timetable
20 %
21 % d(j) > r_d(j)              so -d(j) < -r_d(j)
22 % a(j) > r_a(j)              so -a(j) < -r_a(j)
23 %
24
25 Avec=[(1:n)'       (1:n)'   -ones(n,1) 1*ones(n,1) ;        % d(j) > r_d(
       j)              so -d(j) < -r_d(j)
26       (1:n)'+n     (1:n)'+n -ones(n,1) 1*ones(n,1) ];       % a(j) > r_a(
           j)                 so -a(j) < -r_a(j)
27 Bvec0=[zeros(2*n,1)];                                        % add planned
        arrival time
28 Bvec1=[(1:2*n)'+1];
29 Rvec=[dayplan(:,info.track) ; dayplan(:,info.track)];
30 Cvec=[dayplan(:,info.track) ; dayplan(:,info.track)];
31 pvec=[0;-dayplan(:,info.depart);                                %
       add planned deparue time
32       -dayplan(:,info.arrive)];
33 pnum=[0;(1:n)';(1:n)'];
```

```
34 pname=['emp';ones(2*n,1)*'tbl'];
35
36 %
    ================================================================================

37 %
38 % add running constraints
39 %
40 % a(j) > d(j)+tau_running  so d(j)-a(j) < -tau_running
41 %
42
43 Avec=[Avec;
44       (1:n)'+2*n    (1:n)'     ones(n,1) 2*ones(n,1);          % a(j) > d(j)
          +tau_running  so d(j)-a(j) < -tau_running
45       (1:n)'+2*n    (1:n)'+n -ones(n,1) 2*ones(n,1)];
46 Bvec0=[Bvec0; zeros(n,1)];                                    % add running
    time
47 Bvec1=[Bvec1;(2*n+1:3*n)'+1];
48 Rvec=[Rvec; dayplan(:,info.track)];
49 pvec=[pvec;-dayplan(:,info.run)];
50 pnum=[pnum;(1:n)'];
51 pname=[pname;ones(n,1)*'run'];
52
53 %
    ================================================================================

54 %
55 % add dwell times
56 %
57 % d(j+1) > a(j)+tau_dwell    so   a(j)-d(j+1) < -tau_dwell
58 %
59
60
61 ind=find(dayplan(2:end,end)-dayplan(1:end-1,end)==0);
62 nind=size(ind,1);
63
64 vecitem=3*n+kron((1:nind)',[1;1]);
65 vecarrdep=kron(ind,[1;1])+kron(ones(nind,1),[1;n]);%is plus one because
    of line "ind=" starts at indice 2
66 vecsign=kron(ones(nind,1),[-1;1]);
67 vecctype=3*ones(2*nind,1);
68
69 Avec=[Avec; vecitem vecarrdep vecsign vecctype];
70 Bvec0=[Bvec0;zeros(nind,1)];
71 Bvec1=[Bvec1;(3*n+1:3*n+nind)'+1];
72 Rvec=[Rvec; dayplan(ind,info.track)];
73 pvec=[pvec;-dayplan(ind+1,info.dwell)];
74 pnum=[pnum;ind];
75 pname=[pname;ones(nind,1)*'dwl'];
76
77 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
78
79
80  tela=size(Avec,1)+1;
81  telb=size(Bvec0,1)+1;
82  item=max(Avec(:,1))+1;
83
84
85  %    ================================================================================
86  %
87  % add headway constraints
88  %
89  % d(k) > d(j)+tau_headway(k) + beta*u_l        so   d(j)-d(k) +beta*u_l <
        -tau_headway(k)
90  % d(j) > d(k)+tau_headway(j) + beta*(1-u_l)    so   d(k)-d(j) -beta*u_l <
        -tau_headway(j)-beta
91  % a(k) > a(j)+tau_headway(k) + beta*u_l        so   a(j)-a(k) +beta*u_l <
        -tau_headway(k)
92  % a(j) > a(k)+tau_headway(j) + beta*(1-u_l)    so   a(k)-a(j) -beta*u_l <
        -tau_headway(j)-beta
93  %
94
95  % nruns=max(dayplan(:,info.track)); %number of tracks
96  ell=1;
97  for i=unique(dayplan(:,info.track))'%1:nruns
98      indr=find(dayplan(:,info.track)==i);          % indices track i
99      [~,indrs]=sort(dayplan(indr,info.depart));  % sorted tracks index on
            departure time
100     indr=indr(indrs);                             % dayplan indexes of same
            track ordered on departure time
101     nir=length(indr);
102     dayplan_temp = dayplan(find(dayplan(:,info.track)==i),:);
103     dayplan_temp = sortrows(dayplan_temp,info.depart);
104     length_temp  = size(dayplan_temp,1);
105     for k=2:length_temp
106         for j=1:k-1
107             if (abs((dayplan_temp(k,info.depart)-dayplan(j,info.depart)))
                    >1e-6) ...
108                     &&(abs(dayplan(k,info.depart)-dayplan(j,info.depart))
                            <Nh)
109                 indrj=indr(j);
110                 indrk=indr(k);
111                 Avec(tela:tela+11,:)=[item   indrj    1    4 ;          %
                        d(k) > d(j)+tau_headway +\beta*u_l       so    d(j)-d
                        (k) +beta*u_l < -tau_headway(k)
112                                       item   indrk   -1    4 ;
113                                       item   2*n+ell  beta 4 ;
114                                       item+1 indrk    1    4 ;          %
                                              d(j) > d(k)+tau_headway +\beta
                                             *(1-u_l)   so    d(k)-d(j) -beta
                                             *u_l < -tau_headway(j)-beta
115                                       item+1 indrj   -1    4 ;
```

```
116                                         item+1 2*n+ell -beta 4 ;
117                                         item+2 n+indrj  1    4 ;          %
                                                a(k) > a(j)+tau_headway +\beta*
                                               u_l       so    a(j)-a(k) +beta*
                                               u_l < -tau_headway(k)
118                                         item+2 n+indrk -1    4 ;
119                                         item+2 2*n+ell  beta 4 ;
120                                         item+3 n+indrk  1    4 ;          %
                                                a(j) > a(k)+tau_headway +\beta
                                               *(1-u_l)   so    a(k)-a(j) -beta
                                               *u_l < -tau_headway(j)-beta
121                                         item+3 n+indrj -1    4 ;
122                                         item+3 2*n+ell -beta 4 ];
123
124                 Bvec0(telb:telb+3)  =[-tau_headway;-tau_headway-beta;-
                        tau_headway;-tau_headway-beta];                  % add
                         dwell time
125                 Bvec1(telb:telb+3)  = ones(4,1);
126                 Rvec = [Rvec; [1;1;1;1]*dayplan(indrj,info.track)]; % is
                        toch ook altijd i?
127                 Cvec(2*n+ell,:)=i;
128                 tela=tela+12;
129                 telb=telb+4;
130                 item=item+4;
131                 ell=ell+1;
132
133             end
134         end
135     end
136 end
137 %% make connections
138 [segment_1,segment_2]= find(segment.c_made);
139 transfer_var_table=[];
140 tau_e=cost.transfer;
141 %a=[];
142 for i=1:length(segment_1)
143     %segment 1 and 2
144     id_OC       = segment_1(i);
145     id_CD       = segment_2(i);
146     %find sements indices in dayplan
147     indices_OC  = find(dayplan(:,info.id)==id_OC);
148     indices_CD  = find(dayplan(:,info.id)==id_CD);
149
150     [transfer_labels,orderd_CD]=sortrows(dayplan(indices_CD,:),info.
            depart);
151     indices_CD=indices_CD(orderd_CD);
152     for index_OC = indices_OC'
153         arrive_C=dayplan(index_OC,info.arrive);
154         index_transfer = find(transfer_labels(:,info.depart)>=arrive_C+
                cost.transfer,...
155             1,'first');
156         if ~isempty(index_transfer)
157             Avec = [Avec;...
```

```
158                         item, index_OC+n,                1,5;           %
                               arrival from first train
159                         item, indices_CD(index_transfer), -1,5;     %
                               departure of second train
160                         item, 2*n + ell,                   beta,5];
161                     Bvec0 = [Bvec0; -tau_e];                          % add
                           transfertime
162                     Bvec1 = [Bvec1; 1];
163                     Rvec  = [Rvec; dayplan(index_OC,info.track)];
164                     Cvec  = [Cvec; dayplan(indices_CD(index_transfer),info.
                           track)];
165                     item = item+1;
166                     ell = ell +1;
167                     transfer_var_table=[transfer_var_table;id_OC,id_CD,...
168                         transfer_labels(index_transfer,info.depart),]
169             end
170         end
171     end
172     A=sparse(Avec(:,1),Avec(:,2),Avec(:,3));
173     b=Bvec0+pvec(Bvec1);
174
175     end
```

A-8 Select constraints

The constraints are made using the dayplan; they are over a large time interval, to reduce computation time only relevant constraints and variables are selected. The code "select_constraints.m" selects the relevant constraints and variables in the interval $[t, t+N]$

Code: select_constraints.m

```
1  function [model,var2,bin2,con2,x0_r,c] = select_constraints(t,N,A,dayplan
       ,info,Bvec0,pmat,Bvec1,ub_x,Avec0)
2  %SELECT_CONSTRAINTS Selects the variables and constraint in the interval
       [t,t+N]
3  beta=-600;
4  [nA1,nA2]=size(A);
5  n=size(dayplan,1);
6  nvar=2*n;
7  nbin=nA2-nvar;
8  x0=[dayplan(:,info.depart);dayplan(:,info.arrive);zeros(nbin,1)];
9
10 xold=x0;
11
12
13 %%
14 var.indices = 1:nvar;
15 bin.indices = nvar+1:nvar+nbin;
16 con.indices = 1:nA1;
17 var.deleted = [];
18 bin.deleted = [];
```

```
19 con.deleted = [];
20 [var,bin,con] = remove_indices(var,bin,con,[],[],[]);
21 %% remove constraints of segments departing after t+N
22 remove_var1 = [find(x0(1:n)>t+N); n+find((x0(1:n)>t+N))];    % variables
       of which segment departure>t+N
23 remove_con1 = find(sum(abs(A(:,remove_var1)),2));            %
       constraints with one or more var>t+N
24 remove_bin1 = find(sum(abs(A(remove_con1,bin.indices)),1))'; %bin var in
       removed constraints
25 [var1,bin1,con1] =  ...
26     remove_indices(var,bin,con,remove_var1,remove_bin1,remove_con1);
27 % find(~remove_var1==indvard)
28 % find(~remove_con1==indcond)
29 % find(~remove_bin1+var.length==indbind)
30 %% Remove constraints with no longer needed because in the past
31 clc
32 remove_var2 = find(x0(var1.indices)<=t);
33 [var2,~,~] =  remove_indices(var1,bin1,con1,remove_var2,[],[]);
34 remove_con2 = find(~(sum(abs(A(con1.indices,var2.indices)),2)>0)); %
       constraints with no future variables
35 % todo remove headway constraints if one variable in past
36 remove_bin2 = find(sum(abs(A(con1.indices(remove_con2),bin1.indices)),1))
       ;
37 [~,bin2,con2] =  remove_indices(var1,bin1,con1,[],remove_bin2,remove_con2
       );
38 %%
39 Ared=A(con2.indices,[var2.indices,bin2.indices]);
40 Adel=A(con2.indices,[var2.deleted,bin2.deleted]);
41 b=Bvec0+pmat(Bvec1);
42 bred = b(con2.indices)-Adel*x0([var2.deleted bin2.deleted]); %Adel*x<bred
43 c=[0.01*ones(var.length/2,1);ones(var.length/2,1);zeros(bin.length,1)];
44 c(Avec0(find(Avec0(:,4)==5 & Avec0(:,2)>2*n),2))=0.0001;
45 lb   = x0;
46 ub   = lb + [30 *ones(nvar,1);ub_x*ones(nbin,1)];
47
48 cred=c([var2.indices,bin2.indices]);
49
50 vecCB=['C';67*ones(var2.length-1,1);66*ones(bin2.length,1)]; %difines var
       types, continous or binary
51
52 lb   = lb([var2.indices bin2.indices]);
53 ub   = ub([var2.indices bin2.indices]);
54 %% make model struct for input gurobi
55 model.A     = Ared;
56 model.sense = '<';
57 model.rhs   = bred;
58 model.lb    = lb;
59 model.ub    = ub;
60 model.vtype = vecCB;
61 model.obj   = cred;
62 x0_r        = x0([var2.indices, bin2.indices]);
63 end
```

```
64 function [var,bin,con] = remove_indices(var,bin,con,remove_v,remove_b,
       remove_c)
65 var.deleted           = sort([var.deleted, var.indices(remove_v)]);
66 bin.deleted           = sort([bin.deleted, bin.indices(remove_b)]);
67 con.deleted           = sort([con.deleted, con.indices(remove_c)]);
68 var.indices(remove_v) = [];
69 bin.indices(remove_b) = [];
70 con.indices(remove_c) = [];
71 var.length            = length(var.indices);
72 bin.length            = length(bin.indices);
73 con.length            = length(con.indices);
74 end
```

A-9 Optimazation

The selected constraints and objective are solved with Gurobi in the function "optimization_railway.m".

Code: optimization_railway.m

```
1  function [xopt,stat,crit,critlow] = optimrail(model)
2  %minmize the objective using gurobi
3  params.outputflag = 0;  %Turn off output display
4  params.MIPGap=1e-6;     %Set optimality gap
5  params.outputflag = 0;          % Silence gurobi
6  params.method     = 1;          % Use dual simplex method
7
8  result = gurobi(model, params);
9  stat=result.status;
10 crit=result.objval;
11 critlow=result.objbound;
12 xopt=result.x;
13 end
```

Appendix B

Test case data

This appendix contains all the test case details and results. The first section contains the timetable of the test followed by

B-1 Timedata & stations

This section encloses the timedata and station sheet of the test case described in Section 6-2. This is part of the manifest of the dutch railways of 2011. There is more data on the different train runs which is absent from this table because it is not relevant for the thesis, ask Ton van den Boom for the complete data-file.

Timedata

id	type	Train	track	origin	destination	dep	arr	run	dwell	shift
1	6	4	106	33	23	27	33	5	1	-1
2	6	4	103	23	27	33	41	6	0	0
3	6	4	62	27	11	41	48	6	0	0
4	6	4	58	11	10	48	55	2	0	0
5	6	4	179	10	65	57	64	7	1	0
6	6	4	59	65	32	4	19	13	0	-1
7	6	4	60	32	65	41	55	13	10	0
8	6	4	180	65	10	55	61	6	0	0
9	6	4	57	10	11	4	5	1	2	-1
10	6	4	61	11	27	5	14	6	0	0
11	6	4	104	27	23	14	24	6	0	0
12	6	4	105	23	33	24	32	5	0	0
13	19	13	51	9	10	23	25	2	10	-1

id	type	Train	track	origin	destination	dep	arr	run	dwell	shift
14	19	13	177	10	65	27	33	6	2	0
15	19	13	59	65	32	33	46	12	0	0
16	19	1314	60	32	65	14	28	13	2	-1
17	19	1314	180	65	10	28	34	6	0	0
18	19	1314	52	10	9	35	38	2	1	0
19	19	1314	51	9	10	53	55	2	10	0
20	19	1314	177	10	65	57	61	4	2	0
21	19	1314	59	65	32	1	16	14	0	-1
22	19	14	60	32	65	44	58	13	2	0
23	19	14	180	65	10	58	64	6	0	0
24	19	14	52	10	9	5	8	2	1	-1
25	21	171	97	20	27	14	32	17	2	0
26	21	171	62	27	11	34	42	8	2	0
27	21	171	58	11	10	42	45	1	0	0
28	21	171	177	10	65	48	53	5	2	0
29	21	171	59	65	32	53	67	12	0	0
30	21	172	60	32	65	52	66	13	2	0
31	21	172	180	65	10	6	11	5	0	-1
32	21	172	57	10	11	14	18	1	2	0
33	21	172	61	11	27	18	26	8	0	0
34	21	172	98	27	20	28	46	17	2	0
35	21	181	97	20	27	44	62	17	2	0
36	21	181	62	27	11	4	12	8	2	-1
37	21	181	58	11	10	12	14	1	0	0
38	21	181	177	10	65	18	23	5	2	0
39	21	181	59	65	32	23	38	12	0	0
40	21	182	60	32	65	22	36	13	2	0
41	21	182	180	65	10	36	41	5	0	0
42	21	182	57	10	11	44	48	1	2	0
43	21	182	61	11	27	48	56	8	0	0
44	21	182	98	27	20	58	76	17	2	0
45	22	201	97	20	27	29	46	17	2	-1
46	22	201	62	27	11	49	55	6	2	0
47	22	201	58	11	10	56	60	1	1	0
48	22	201	177	10	65	3	9	6	2	-1
49	22	201	59	65	32	9	22	12	0	0
50	22	202	60	32	65	37	51	12	2	-1
51	22	202	180	65	10	51	57	6	0	0
52	22	202	57	10	11	59	60	1	2	0
53	22	202	61	11	27	1	10	7	1	-1
54	22	202	98	27	20	13	31	17	2	0
55	22	211	97	20	27	59	76	17	2	0
56	22	211	62	27	11	19	25	6	2	-1
57	22	211	58	11	10	26	30	1	1	0
58	22	211	177	10	65	33	39	6	2	0

id	type	Train	track	origin	destination	dep	arr	run	dwell	shift
59	22	211	59	65	32	39	52	12	0	0
60	22	212	60	32	65	7	21	12	2	-1
61	22	212	180	65	10	21	27	6	0	0
62	22	212	57	10	11	29	30	1	2	0
63	22	212	61	11	27	31	40	7	1	0
64	22	212	98	27	20	43	61	17	2	0
65	26	22	106	33	23	37	39	2	1	0
66	26	22	103	23	27	39	53	12	0	0
67	26	22	64	27	11	55	63	5	1	0
68	26	22	54	11	9	3	7	2	0	-1
69	26	22	53	9	11	23	24	1	10	0
70	26	22	61	11	27	24	30	5	0	0
71	26	22	104	27	23	37	50	12	2	0
72	26	22	105	23	33	50	52	2	0	0
73	26	23	106	33	23	7	9	2	1	-1
74	26	23	103	23	27	9	22	12	0	0
75	26	23	64	27	11	25	33	5	1	0
76	26	23	54	11	9	33	37	2	0	0
77	26	23	53	9	11	53	54	1	10	0
78	26	23	61	11	27	54	60	5	0	0
79	26	23	104	27	23	7	20	12	2	-1
80	26	23	105	23	33	20	22	2	0	0
81	37	39	53	9	11	8	9	1	10	-1
82	37	39	61	11	27	9	20	7	0	0
83	37	39	104	27	23	22	29	7	2	0
84	37	39	105	23	33	29	37	6	0	0
85	37	41	106	33	23	52	58	6	1	0
86	37	41	103	23	27	58	68	7	0	0
87	37	41	64	27	11	10	18	5	1	-1
88	37	41	54	11	9	18	22	2	0	0
89	37	41	53	9	11	38	39	1	10	0
90	37	41	61	11	27	39	50	7	0	0
91	37	41	104	27	23	52	59	7	2	0
92	37	41	105	23	33	59	67	6	0	0
93	37	43	106	33	23	22	28	6	1	0
94	37	43	103	23	27	28	38	7	0	0
95	37	43	64	27	11	40	48	5	1	0
96	37	43	54	11	9	48	52	2	0	0
97	43	43	53	9	11	11	12	1	10	-1
98	43	43	63	11	27	13	28	10	1	0
99	43	43	104	27	23	30	47	17	2	0
100	43	43	189	23	33	51	55	2	1	0
101	43	35	190	33	23	34	40	4	1	0
102	43	35	103	23	27	42	59	16	1	0
103	43	35	64	27	11	2	14	10	1	-1

id	type	Train	track	origin	destination	dep	arr	run	dwell	shift
104	43	35	54	11	9	15	19	2	1	0
105	43	35	53	9	11	41	42	1	10	0
106	43	35	63	11	27	43	58	10	1	0
107	43	35	104	27	23	0	17	16	2	-1
108	43	35	189	23	33	21	25	4	1	0
109	43	35	190	33	23	4	9	4	1	-1
110	43	35	103	23	27	12	29	16	1	0
111	43	35	64	27	11	32	44	10	1	0
112	43	35	54	11	9	45	49	2	1	0
113	50	46	51	9	10	43	47	2	10	0
114	50	46	179	10	65	48	57	9	1	0
115	50	46	59	65	32	57	73	14	0	0
116	50	46	60	32	65	17	32	14	1	-1
117	50	46	178	65	10	32	44	12	0	0
118	50	46	52	10	9	45	48	2	1	0
119	50	46	51	9	10	13	17	2	10	-1
120	50	46	179	10	65	18	27	9	1	0
121	50	46	59	65	32	27	43	14	0	0
122	50	46	60	32	65	47	62	14	1	0
123	50	46	178	65	10	2	14	12	0	-1
124	50	46	52	10	9	15	18	2	1	0
125	51	681	51	9	10	58	62	2	10	0
126	51	681	179	10	65	3	12	9	1	-1
127	51	681	59	65	32	12	28	14	0	0
128	51	682	60	32	65	32	47	14	2	0
129	51	682	178	65	10	47	59	12	0	0
130	51	682	52	10	9	0	3	2	1	-1
131	51	691	51	9	10	28	32	2	10	0
132	51	691	179	10	65	33	42	9	1	0
133	51	691	59	65	32	42	58	14	0	0
134	51	692	60	32	65	2	17	14	2	-1
135	51	692	178	65	10	17	29	12	0	0
136	51	692	52	10	9	30	33	2	1	0
137	63	85	97	20	27	52	73	18	10	-1
138	63	85	64	27	11	17	29	11	2	-1
139	63	85	54	11	9	30	34	2	1	0
140	63	85	53	9	11	56	57	1	10	0
141	63	85	63	11	27	58	73	11	1	0
142	63	85	98	27	20	16	38	19	2	-1
143	63	86	97	20	27	22	43	18	10	-1
144	63	86	64	27	11	47	59	11	2	0
145	63	86	54	11	9	0	4	2	1	-1
146	63	86	53	9	11	26	27	1	10	0
147	63	86	63	11	27	28	43	11	1	0
148	63	86	98	27	20	46	68	19	2	0

id	type	Train	track	origin	destination	dep	arr	run	dwell	shift
149	92	921	106	33	23	1	5	4	2	-1
150	92	921	103	23	27	6	16	8	0	0
151	92	921	62	27	11	16	22	6	0	0
152	92	921	58	11	10	22	25	2	0	0
153	92	921	179	10	65	27	36	4	2	0
154	92	921	59	65	32	36	49	11	0	0
155	92	922	60	32	65	11	25	13	2	-1
156	92	922	180	65	10	25	31	6	0	0
157	92	922	57	10	11	34	35	1	2	0
158	92	922	61	11	27	35	44	9	0	0
159	92	922	104	27	23	44	54	7	0	0
160	92	922	105	23	33	54	57	3	0	0

Station

id	name	id	name	id	name
1	Alkmaar	23	Hoofddorp	45	Heerhugowaard
2	Amersfoort	24	Hoorn	46	Breukelen
3	AmsterdamC	25	LageZwaluwe	47	Almere
4	Arnhem	26	Leeuwarden	48	Lelystad
5	Boxtel	27	Leiden	49	Hilversum
6	Breda	28	Maastricht	50	AlphenadRijn
7	Den Helder	29	Meppel	51	Maarn
8	DenBosch	30	Nijmegen	52	Geldermalsen
9	DenHaagCS	31	Roosendaal	53	Den Dolder
10	DenHaagHS	32	Rotterdam	54	Baarn
11	DenHaagNOI	33	Schiphol	55	Geldermalsen
12	Deventer	34	Sittard	56	Tiel
13	Dordrecht	35	Tilburg	57	Weert
14	Duivend	36	Uitgeest	58	Roermond
15	Eindhoven	37	Utrecht	59	Rhenen
16	Enschede	38	Venlo	60	Maastricht R
17	Goes	39	Vlissingen	61	Apeldoorn
18	Gouda	40	Weesp	62	Zutphen
19	Groningen	41	Woerden	63	EdeWageningen
20	Haarlem	42	Zaandam	64	Deurne
21	Heerlen	43	Zwolle	65	Delft Aansl
22	Almelo	44	Enkhuizen	66	Gouda Goverw

Bibliography

[1] Van Der Wal J. Al-Ibrahim A. Association for European Transport and contributors 2005. *Transport*, 5:1–17, 2005.

[2] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route planning in transportation networks*, volume 9220 LNCS. 2016.

[3] Hannah Bast and Sabine Storandt. Frequency-based search for public transit. *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 04-07-Nove:13–22, 2014.

[4] Annabell Berger, Christian Blaar, Andreas Gebhardt, Matthias Müller-Hannemann, and Mathias Schnee. Passenger flow-Oriented train disposition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6942 LNCS:227–238, 2011.

[5] Claus Biederbick and Leena Suhl. Decision support tools for customer-oriented dispatching. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4359 LNCS:171–183, 2007.

[6] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014.

[7] Graziana Cavone, Virginia Montaruli, Ton J.J. Van Den Boom, and Mariagrazia Dotoli. Demand-Oriented Rescheduling of Railway Traffic in Case of Delays. *7th International Conference on Control, Decision and Information Technologies, CoDIT 2020*, pages 1040–1045, 2020.

[8] Brian C. Dean. Continuous-Time Dynamic Shortest Path Algorithms. *Electrical Engineering*, page 116, 1999.

[9] Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. *Robust and online large-scale optimization*, 2:207–230, 2009.

[10] E W Dijkstra. A Note on T w o Problems in Connexion with Graphs. 271:269–271, 1959.

[11] Twan Dollevoet, Dennis Huisman, Leo Kroon, Marie Schmidt, and Anita Schöbel. Delay management including capacities of stations. *Transportation Science*, 49(2):185–203, 2015.

[12] Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay management with re-routing of passengers. *OpenAccess Series in Informatics*, 12(March 2021), 2009.

[13] Michel Gendreau, Gianpaolo Ghiani, and Emanuela Guerriero. Time-dependent routing problems: A review. *Computers and Operations Research*, 64:189–197, 2015.

[14] Nadjla Ghaemi, Oded Cats, and Rob M.P. Goverde. Railway disruption management challenges and possible solution directions. *Public Transport*, 9(1-2):343–364, 2017.

[15] Eric Halsey. The Shortest Route Through a Network with Time-Dependent Internodal Transit Times. pages 493–498, 1966.

[16] Pavle Kecman, Francesco Corman, Andrea D'Ariano, and Rob M.P. Goverde. Rescheduling models for railway traffic management in large-scale networks. *Public Transport*, 5(1-2):95–123, 2013.

[17] Bart Kersbergen, János Rudan, Ton van den Boom, and Bart De Schutter. *Towards railway traffic management using switching Max-plus-linear systems: Structure analysis and rescheduling*, volume 26. 2016.

[18] Bart Kersbergen, Ton van den Boom, and Bart De Schutter. Distributed model predictive control for railway traffic management. *Transportation Research Part C: Emerging Technologies*, 68:462–489, 2016.

[19] Eva König. A review on railway delay management. *Public Transport*, 12(2):335–361, 2020.

[20] Eva König and Cornelia Schön. Railway delay management with passenger rerouting considering train capacity constraints. *European Journal of Operational Research*, 288(2):450–465, 2021.

[21] Richard M. Lusby, Jesper Larsen, and Simon Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266(1):1–15, 2018.

[22] Matthias Müller-Hannemann and Ralf Rückert. Dynamic Event-Activity Networks in Public Transportation. *Datenbank-Spektrum*, 17(2):131–137, 2017.

[23] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Experimental comparison of shortest path approaches for timetable information. *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithms and Combinatorics*, (January):88–99, 2004.

[24] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12(August 2014), 2008.

[25] Danny Schipper and Lasse Gerrits. Differences and similarities in European railway disruption management practices. *Journal of Rail Transport Planning and Management*, 8(1):42–55, 2018.

[26] Anita Schöbel. A model for the delay management problem based on mixed-integer-programming. *Electronic Notes in Theoretical Computer Science*, 50(1):1–10, 2001.

[27] Anita Schöbel. Capacity constraints in delay management. *Public Transport*, 1(2):135–154, 2009.

[28] Leena Suhl and Taïeb Mellouli. Requirement for, and Design of, an Operations Control System for Railways. pages 371–390, 1999.

[29] N. J.A. Van Exel and P. Rietveld. Could you also have made this trip by another mode? An investigation of perceived travel possibilities of car and train travellers on the main travel corridors to the city of Amsterdam, The Netherlands. *Transportation Research Part A: Policy and Practice*, 43(4):374–385, 2009.

[30] Mark Van Hagen. *Waiting Experience at Train Stations*. 2011.

Glossary

List of Acronyms

APSPA	All-Pair Shortest Path Algorithm
AW	Always Wait
DM	Delay Management
DSS	Decision Support System
EAN	Event Activity Network
FIFO	First In First Out
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Non-Linear Programming
NS	Dutch Railways (Dutch: Nederlandse Spoorwegen)
NW	No Wait
OD	Origin-Destination
SPA	Shortest Path Algorithm
RTR	Real-Time Rescheduling
TAD	Train Service Handling Document (Dutch: Treindienst Afhandelings Document)
TTF	Travel Time Function
TD	Train Dispatcher

Index