Resolving infeasibilities in the PESP model of the Dutch railway timetabling problem

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

GERT-JAAP POLINDER

Delft, the Netherlands
August 2015

Copyright © 2015 by Gert-Jaap Polinder. All rights reserved.
MSc THESIS APPLIED MATHEMATICS

“Resolving infeasibilities in the PESP model of the Dutch railway timetabling problem”

Gerrit Jacob Polinder

Delft University of Technology

Daily supervisor
Prof. dr. L.G. Kroon

Responsible professor
Prof. dr. ir. K.I. Aardal

Other thesis committee members
Dr. M. Molinaro

August 2015
Delft, the Netherlands
Preface

In this master thesis the results of the final research I did as part of my studies in Applied Mathematics is described. It is written to obtain a masters degree in Applied Mathematics and marks the end of the period as a student at Delft University of Technology. This research clearly shows what applied mathematics is about. What for many people seems to be a dull subject, is very useful in practice. This is only one example, many more can be given.

From the beginning of my studies, I was interested in logistics and trains. When the time was there to look for a thesis project, I came in contact with Leo Kroon from Rotterdam School of Management, Erasmus University Rotterdam, thanks to my supervisor Karen Aardal. Professor Kroon has been doing research on railway related problems for a long time and has been involved in the team that received the Franz Edelman Award for their work on train scheduling problems. Leo has introduced me at the Innovation Department of Netherlands Railways to carry out my masters research there. In this research, I have developed a method that resolves infeasibilities in the railway timetabling problem.

During this research, I got the help of quite a few people. First of all, I would like to thank my colleagues from NS for their help on my research. A few have to be mentioned by name. First of all, my thanks go to Gábor. Although there is no free lunch, you gave many useful advices how to get the ‘most cheapest’ that is possible. Next, thanks go to Joël, Pieter-Jan and Leon for helping in a lot of DONS-related issues. Further, I would like to thank Stefan Schuurman (ORTEC) for his advices and answers to a lot of questions on the CADANS software.

Most important, I would like to thank my supervisors for their help on this thesis, Leo, Karen and Marco. Many thanks for your comments, your help and your patience during this project. It was a lot of fun working on this research under your supervision. Leo, many thanks for introducing me to the fascination world of optimisation in public transport. A lot of interesting problems arise in this field and still a lot of research can be done in this field, I am looking forward to that. Karen and Marco, many thanks for your comments and advices during this research project. You clearly showed me that things that became more or less obvious for me are not that obvious at all and are challenging to be explained in a clear way.

Last but not least, many thanks to my wife Chantine. Thanks for listening to all my ‘very interesting’ train facts and my programming issues. You put some constraints to my pastime optimisation model that were very relevant, also to the success of this thesis. If those constraints were violated any time as well, you turned my attention to keep me on track, thanks!

One final note to all of you: I hope you enjoy reading this thesis as much as I did working on it, have fun!

Gert-Jaap Polinder

Rotterdam, August 2015
Contents

Preface i

Table of contents v

List of Figures v

List of Tables viii

1 Introduction 1

1.1 A description of the Dutch railway network 3
1.2 The need for timetabling 4
1.3 The planning process in practice 5
1.4 Designing a timetable 6

1.4.1 Time space diagram 6
1.4.2 Platform occupation chart 6
1.4.3 Timetable planning 9

1.5 Problem description 9

1.5.1 Conflicts 10
1.5.2 Problem statement 13

1.6 Literature review 13
1.7 Outline of the thesis 14

2 Optimisation techniques 15

2.1 Computational Complexity 15
2.2 Constraint programming 16
2.3 Mixed Integer Linear Programming 16

2.3.1 Linear Programming 16
2.3.2 Integer Linear Programming 17
2.3.3 Mixed Integer Linear Programming 19
2.3.4 Branch and cut 20

3 The model and the current solver 23

3.1 Introduction 23
3.2 The current model 24

3.2.1 Trip time and dwell time constraints 27
3.2.2 Frequency and synchronisation constraints 27
3.2.3 Fixed arrival or departure times 28
3.2.4 Safety constraints: headway 29
3.2.5 Safety constraints: incoming-outgoing and collision 30
3.2.6 Safety constraint: overtaking 32
3.2.7 Safety constraints: hindrance 32
3.2.8 Connection constraints 35

3.3 Properties of the model 37

3.3.1 Constraint graph 37
3.3.2 Reducing the problem 37
3.3.3 If the p-values are given 38
3.3.4 Safety clique 39
3.3.5 Cycles 39
List of Figures

1.1 Country Performance on RPI .......................................................... 2
1.2 RPI Ratings correlate with Public Cost .......................................... 2
1.3 Allocating Public Subsidies to Infrastructure Managers Generates Greater Value ........................................................................ 3
1.4 Stages of the planning process ......................................................... 5
1.5 Basis Hourly Pattern: Rotterdam - Utrecht .......................................... 7
1.6 Platform Occupation Chart (Gouda, NS timetable 2009) ..................... 8
1.7 Different use of the same track .......................................................... 10
1.8 Route of the 500 and 1700 series ....................................................... 11
1.9 Several types of intercities in use at NS ............................................. 12
1.10 Uitbug-situation ........................................................................... 13

2.1 Feasible region example LP .............................................................. 18
2.2 Feasible region example IP ............................................................... 18
2.3 Feasible region example IP ............................................................... 19
2.4 Branch and bound tree for the example problem ............................. 21

3.1 Outline DONS .............................................................................. 23
3.2 Two train path points ...................................................................... 24
3.3 The railway safety block system ...................................................... 29
3.4 Out-out and in-in relations .............................................................. 30
3.5 Out-in and in-out relations .............................................................. 31
3.6 Trains overtaking each other ............................................................ 32
3.7 Crossing trainpaths at Barneveld ..................................................... 33
3.8 Fly-over at Harmelen .................................................................. 33
3.9 Crossing trainpaths in The Hague ................................................... 34
3.10 Flexible 2-2 connections ................................................................. 35
3.11 Changing and/or constraints ........................................................... 36
3.12 A constraint in the graph ............................................................... 37
3.13 Trip and dwell arcs ...................................................................... 37
3.14 Merged arcs ................................................................................ 38
3.15 Parallel arcs ................................................................................ 38
3.16 Small cycle for two trains on a track .............................................. 39
3.17 Example of a graph that has no solution ....................................... 42
3.18 Simplified cycles ......................................................................... 43
3.19 Two trains share a track ................................................................. 44
3.20 An arc with $l_{a} = u_{a}$ can be contracted ........................................ 46
3.21 A node of degree two can be removed ............................................. 46
3.22 Triangle propagation .................................................................. 46

4.1 Overview FB-situation .................................................................. 54
4.2 Overview connection situation ......................................................... 55
4.3 Example of a situation showing that a conflict might contain less arcs 57

B.1 Original graph ................................................................................ 81
B.2 Removing three arrival nodes .......................................................... 81
B.3 Removing last arrival nodes ........................................................... 81
B.4 Removing an arc with $l = u$ ............................................................ 82
B.5 Merging two arcs into one ............................................................... 82
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.6</td>
<td>Removing an arc with $l = u$</td>
<td>82</td>
</tr>
<tr>
<td>B.7</td>
<td>Final reduced graph</td>
<td>83</td>
</tr>
<tr>
<td>B.8</td>
<td>A possible solution</td>
<td>83</td>
</tr>
<tr>
<td>D.1</td>
<td>Overview of the network</td>
<td>99</td>
</tr>
<tr>
<td>D.2</td>
<td>Situation around Dv</td>
<td>102</td>
</tr>
<tr>
<td>D.3</td>
<td>Time space diagram for Amf-Zl</td>
<td>104</td>
</tr>
<tr>
<td>D.4</td>
<td>Time space diagram for Zl-Dv</td>
<td>105</td>
</tr>
<tr>
<td>D.5</td>
<td>Time space diagram for Amf-Dv</td>
<td>105</td>
</tr>
<tr>
<td>E.1</td>
<td>Overview of the network between Sgn-Hdr</td>
<td>108</td>
</tr>
</tbody>
</table>
List of Tables

5.1 Results onHdr30 ......................................................... 67
5.2 Summary of results on NL13, algorithm 4.3 .................................. 67
5.3 Summary of results on NL13, algorithm 4.4 .................................. 68
5.4 Summary of results on NL13new, algorithm 4.3 ................................. 68
5.5 Summary of results on NL13new, algorithm 4.4 ................................. 69
A.1 Abbreviations for nodes on the network ............................................. 78
D.1 Line plan for the triangular network ................................................ 100
E.1 Trip times for the two series ............................................................ 108
Chapter 1

Introduction

One of the most important modes of public transport in the Netherlands is the train. On a daily basis, more than a million passengers take the train, spread out over more than 4500 train services. According to a study carried out by ‘Centraal Bureau voor de Statistiek’ (CBS (2009)), based on statistics from 2006, the Netherlands has the most heavily used rail network in the European Union. Cities are close to each other so commuting is very common. In Europe, only the network of Switzerland is used more intensively. This heavy use is due to the fact that the Netherlands has very few rail kilometres per inhabitant, 17 kilometres of network-length per 100,000 inhabitants, while the average in the European Union is 44. A rail kilometre is one kilometre of single track. If there are two parallel tracks, 500 metres of double track leads to one rail kilometre.

The distance a train travels is measured in train kilometres, which is the number of train services multiplied by the length of the service in kilometres. Given this definition, the Dutch railway network has more than 20,000 train kilometres per kilometre of track per year, which is more than twice the European average. Per kilometre of track, there are almost 2.5 million passenger kilometres per year, which is the number of passengers multiplied by the distance they travel, in the Netherlands, while the European average is just above one million. For both measures, the Dutch railway network scores as the most intensively used network in the European Union.

About 92% of the train kilometres are made by passenger trains. Although there is freight transport by rail, especially on corridors like the ‘Betuweroute’, the Netherlands do not play an important role when compared to other countries in the European Union. When the number of tons of goods is calculated per kilometre of track, the Netherlands scores just below the European average. This is due to the fact that transporting goods by rail is only profitable when it is transported over a long distance and inland shipping is an excellent alternative in the Netherlands, which other countries do not have.

In 2015, the Boston Consulting Group did research on the quality of the rail systems in Europe. For each country, a Rail Performance Index (RPI) was calculated, which was a weighted sum of the scores on safety, quality of service and intensity of use. The results are summarised in Figure 1.1. It can be seen that the Netherlands scores at position 10 out of 25. Explanations that are mentioned in Boston Consulting Group (2015) for this score are the following:

1. The Netherlands score high on the safety aspect.
2. The intensity of use is limited since passenger and freight transport are both weighted at 50%. Since freight transport in the Netherlands is rather limited, the score is rather low.
3. The score on quality of service is poor.

Those scores show that especially concerning passenger transport the intensity of use is rather high. Further, the quality of service can be improved. One of the conclusions that was drawn in the report is that the RPI ratings correlate with Public Cost. However, some countries get more value for their money. In Figure 1.2, a graph is shown in which the Public Cost are shown against the RPI rating.

Furthermore, not every country allocates the money they spend on the rail system to the same parties. Some spend more on the infrastructure managers, others more on the transporting companies. They found that countries in which the public subsidies focus on the infrastructure.

1English: Statistics Netherlands
CHAPTER 1. INTRODUCTION

Figure 1.1: Country Performance on RPI

Figure 1.2: RPI Ratings correlate with Public Cost
1.1. A DESCRIPTION OF THE DUTCH RAILWAY NETWORK

A graph in which the percentage of public cost that goes to the infrastructure manager are shown against the value for money is shown in Figure 1.3.

### 1.1 A description of the Dutch railway network

In the year 1839, the first railway line came into operation between Amsterdam and Haarlem. Soon different companies added new lines and exploited them. Over the years, several companies merged and in 1937 the ‘Nederlandse Spoorwegen’ (NS) was founded. Until the nineties, they operated almost all lines in the Netherlands, except some international lines. Initially most lines functioned more or less separately and were not fully integrated into one coherent network. This slowly changed, but planning the lines as a coherent network was done for the first time around 1970.

Currently, there are several companies present on the Dutch railroads offering public transport services. For a long time, NS was the only provider of public transport by rail, but due to regulations it is made possible for other companies to exploit lines. Although NS is still the largest operator on the rail network, it operates almost all intercity lines and many regional lines, a few contestants are present as well. Arriva, a subsidiary of Deutsche Bahn (DB), exploits some lines, mainly in the eastern and northern part of the country, and companies like Syntus, Veolia and Connexxion serve a few lines in the eastern part of the country as well. NS is also present in other countries with its subsidiary Abellio. Until now, NS got the right to exploit the intercity network from the government without a tender. In this network, the regional trains that share their tracks are included as well. Only at the outskirts of the network, there is interaction with the other companies.

Concerning freight transport by rail, there are some corridors dedicated to freight transport, like the ‘Betuweroute’, a line from the Maasvlakte (Rotterdam) to the border with Germany at Zevenaar. However, many freight trains use, at least for a part of their route, the same tracks as the passenger trains do. Since they cannot only travel at night, they interact with the passenger managers, tend to get a higher performance at a lower cost per capita than other countries. A graph in which the percentage of public cost that goes to the infrastructure manager are shown against the value for money is shown in Figure 1.3.

---

2 English: Netherlands Railways.
3 For a full overview of the different rail companies and the lines they serve, the interested reader is referred to: [https://en.wikipedia.org/wiki/Rail_transport_in_the_Netherlands](https://en.wikipedia.org/wiki/Rail_transport_in_the_Netherlands) (consulted on August 18, 2015)
trains. Such interactions between trains of different companies give rise to conflicts in the planning process.

In 1995, NS became an independent organisation. The management of the infrastructure went to the organisations NS Railinfrabeheer\(^4\), NS verkeersleiding\(^5\), and Railned, although NS was still the owner of the infrastructure until 2002. From then on, those three organisations merged into ProRail, which became the manger of the infrastructure. Nowadays, ProRail is the company that grants permission to operators to drive on their tracks. They give each train operator a certain capacity and timeslots in which they may drive their trains and ask a compensation for that. Therefore, companies have to request track capacity on the network from ProRail. They can provide an initial timetable for their part of the network, and ProRail has to combine them all into one final timetable. This means that the companies have to negotiate together and with ProRail about their timeslots in the railway network. Since NS is the largest operator on the network, they have a lot of influence on the timetable. Beside the passenger trains, there has to be capacity for the freight trains as well. Since they are quite slow and use a lot of time to accelerate, they consume a lot of space and time of the network.

Sometimes maintenance of infrastructure is taken into account in timetabling. This only concerns projects that are planned long time in advance. However, if it is taken into account, this mostly concerns the assignment of trains to platforms at the stations. Some time ago, the station of Rotterdam has been redesigned. During this process, trains were assigned another track when the building process moved from one platform to the other. It actually never happens that part of the infrastructure is unavailable during a whole year, that would be too expensive. Therefore, track maintenance can not really be incorporated into the timetable. Most of this work is done at night or in weekends. Further, sometimes infrastructure can not be made available during some period of time, but it is not integrated into the timetabling and concerns temporary timetable changes only.

1.2 The need for timetabling

Around the sixties, NS began to make less money. The main reason for this was the decrease in the transport of coal\(^6\) and the growing popularity of the car at the expense of the train. Wages of rail road personnel were low already, so they could not be decreased any further. Although closing small lines might seem to be a locally optimal solution, those lines served as feeders for other lines. This would lead to less passengers on the full network, so it would not be an optimal solution. Therefore NS had to take another type of action, which they did by introducing the plan Spoorslag ’70 in order to make the train more competitive. The most important part of this plan was a new timetable that came into operation in December 1969. The main characteristics of this new timetable were:

- It became a periodic timetable. This means that if a train from Utrecht to Rotterdam leaves at 12.05, it will leave every hour at 5 minutes past. The timetable repeats itself every hour.

- Better connections. The new timetable focuses more on the large stations in the network. At those places, there should be good connections between trains in order to decrease travel time of passengers.

- Symmetric timetable. In the new timetable, travel times in both directions should be (more or less) equal. Also, the same connections should be made possible.

- A new network of lines was designed. The trains stopped always at the same stations, which was to some extent variable in the past.

- Frequencies increased, to provide more flexibility to the customers.

Few adaptations in infrastructure were needed to operate this timetable. This plan worked out very well, since passenger numbers increased greatly. Currently, railway timetables in the Netherlands still satisfy those property. It is a coherent network of train lines, having many connections between

\(^1\)Management of infrastructure
\(^2\)English: traffic management
\(^6\)DB Cargo and NS Cargo were the freight transport of the German and Dutch Railways. In 2000, they merged into Railion, currently called DB Schenker Rail.
train lines. Next to this, it still is as symmetric as possible, which should make it easy for customers. If they have some connection on their trip in one direction, they have the same connection on the trip in the backward direction. Since 1970, frequencies have still increased. Around that time, the lines were served at most two times per hour. Nowadays, many lines are served four times per hour and plans are to increase this to six or eight times per hour.

1.3 The planning process in practice

The planning process consists of several stages, especially when designing a timetable from scratch and not based on that of the previous year. Schematically this is shown in Figure 1.4.

First of all, the expected number of passengers at the various parts of the country is estimated. This leads to having an Origin-Destination (OD) matrix. Based on this, the line planning can be done. Examples of approaches to the line planning problem are shown in Borndörfer, Grötschel, and Pietsch (2007); Bussieck, Kreuzer, and Zimmermann (1997); Claessens, van Dijk, and Zwaneveld (1998); Goossens, van Hoesel, and Kroon (2004); Schöbel and Scholl (2006). Many of them use heuristic methods to solve these problems, resulting in solutions of great quality. Many, if not all, of those problems are NP-complete, so solving them to optimality might be very hard if the problem size is large, as they are in practice.

When the line plan is known, a timetable can be constructed for this line plan. This is also done in several stages. First of all, an hourly pattern is made. The process to do so is described extensively in this thesis. Once having a basic hourly pattern, a weekly timetable can be made. This incorporates the changes that might happen over the year. Examples of such changes are large maintenance periods and holiday periods. Typically, the daily timetable is made very close to the considered day in order to incorporate possible maintenance and other works on the infrastructure. For the real operation, the management for planning is given to the operations department, that takes care of changes on the day itself, for example due to accidents.

Rolling stock is the most expensive part of the operations in the rail sector. Rolling stock concerns the actual train units driving. Crew costs are about one third of the rolling stock costs. Therefore, first the rolling stock is scheduled. This can be done by solving large Mixed Integer Programming models. As is the case with timetabling as well, the models for the rail sector are much larger and more complicated than those for bus networks. Examples of creating the schedules of the rolling stock in rail networks can be found in Abbink, van den Berg, Kroon, and Salomon (2004); Lin and Kwan (2014).

Once having the rolling stock schedule, the crew scheduling is done. This means assigning duties to trained personnel. At NS, duties are divided between groups of crew members. Those groups get duties assigned and they can decide within each group who actually performs a certain duty. A case study of crew scheduling optimisation can be found in Kwan (2011). In Abbink et al. (2010), a practical example is shown for NS how to perform the large scale crew scheduling in practice. Those problems are often solved using column generation.

It is possible to do the rolling stock and crew scheduling simultaneously, but since the rolling stock costs are much higher than the crew costs, this is not beneficial for rail companies. Huisman (2004) shows how the scheduling of personnel and rolling stock can be integrated, which is done especially at bus companies.
1.4 Designing a timetable

When a timetable is developed, a point in time is found for each train-path point on the network. These are points where conflicts can occur, like stations, junctions or places where tracks merge or split, more on this in Section 3.2. In the planning processes, these points are not called by their full name, but are abbreviated. Throughout this thesis, some of them will be used as well, especially in equations and graphs. A list of abbreviations used can be found in Appendix A.

There are two important graphs that are used in the planning process of a timetable. The first of them is a time-space diagram, the second is a platform occupation chart.

1.4.1 Time space diagram

The time space diagram graphically shows the movements of train lines on a certain part of the network. One axis represents space, the other represents time. In Figure 1.5 a time-space diagram is shown for the part between Rotterdam and Utrecht. As an extension to a standard time-space diagram, the number of tracks are shown at the bottom of the figure. If multiple tracks are available, trains can pass each other, corresponding to crossing lines in the graph. Further, in this diagram, the vertical axis represents time and the horizontal axis represents space, corresponding to the part of the network Rtd-Ut. In the diagram shown, there are two trains (line 130) travelling between Rotterdam (Rtd) and Utrecht (Ut), and two trains (line 100) travelling between The Hague and Utrecht. Those lines share tracks between Hgwbr (a point just before Gouda (Gd)) and Utrecht. When the train stops at a station, a vertical line is shown in the graph, since time moves on while the train stays at the same place. There are horizontal parts as well, meaning time stands still while the position of the train keeps changing. Of course, time moves on if this position changes, but this is due to the rounding to minutes. Sometimes, two train path points might be very close to each other, meaning it takes only a few seconds to travel from the first to the second. Hence, the moment in time those points are passed are equal. One of the trains travelling from Utrecht to Rotterdam is highlighted. It can be observed that the train leaves Utrecht at .15 and arrives in Rotterdam at .51. Also, the points in time it passes the several train path points are shown. The tracks it uses are shown in red.

It can be noticed that the trains coming from The Hague and from Rotterdam are synchronised at Gouda. This means that trains drive with a frequency of four between Gd and Ut, i.e. every 15 minutes a train leaves. The frequency on the parts Gvc-Gd and Rtd-Gd is two, so every 30 minutes a train leaves Rtd (line 130) and Gvc (line 100).

A time-space diagram clarifies which trains are at which point in time on the network. It shows when trains are close to each other and might cause delay in the operation. Planners also use this graph to estimate if it is possible to add an extra train.

1.4.2 Platform occupation chart

Although there might be enough capacity on the tracks between stations, this does not necessarily mean that enough capacity is available at the stations itself. From 1970 onwards, a focus is put on the connections between trains in designing a timetable. In order to have the best connections, many trains need to be at a station at the same time. However, often there is not enough capacity at the stations to make this possible. In order to do the planning at stations, planners use a Platform Occupation Chart. An example of such a chart is shown in Figure 1.6.

In the graph, the horizontal axis corresponds to time. The different tracks present at a station are shown one above the other, so the vertical axis corresponds to space. The Platform Occupation Chart for Gouda of the timetable in 2009 is shown in Figure 1.6. It clearly shows that the station of Gouda has 11 tracks.

At the left in the Figure, the platforms are shown. There are platforms just above track 11, between tracks 8-10 and between tracks 3-5. Gouda also has the situation in which tracks end in the station, so trains can enter it only from one side. This situation is at tracks 4 and 9.

At most stations, tracks have an A- and a B-side, the A side corresponds to the upper side of the line in the graph corresponding to a track, the B-side corresponds to the lower side. For example, it shows that train D2800, coming from Rotterdam (Rtd), enters the station at .07 and
Figure 1.5: Basis Hourly Pattern: Rotterdam - Utrecht
Figure 1.6: Platform Occupation Chart (Gouda, NS timetable 2009)
occupies track 3, both the A- and the B-side. It leaves at .08 for Amersfoort Schothorst (Amfs).
Train C19800, coming from The Hague Central Station (Gvc) and heading for Gouda Goverwelle (Gdg),
passes the B-side and stops at the A-side of platform 5 at .04 and leaves again at .05. When
a train just passes a station, it is shown by a vertical line, since they enter and leave in the same
minute. When possible, this is done on tracks that have no platform next to it, like tracks 2 and
7 at Gouda, in order to avoid using limited capacity. A train turning at Gouda is shown at track
9. This is the train coming from Alphen aan de Rijn (Apn), it stays at Gouda from .09 until .21,
and goes back again.

In this diagram, it is also visible which connections between trains are possible. Suppose you
arrive with the intercity from Groningen (Gn) at .04 and you want to go to Rotterdam. Then
you have to change at Gouda since the train from Groningen goes to The Hague Central Station
(Gvc). The train from Groningen arrives at platform 10, and a few minutes later (at .08), a train
to Rotterdam arrives at platform 8. So the connection between those intercities is a cross-platform
connection.

1.4.3 Timetable planning

In 1970, planning the timetable was a process that was done by hand. No automatic tools were
available and because of the low frequencies (trains had a frequency of at most twice per hour,
while currently some lines drive six times per hour) it was still possible to do the planning by
hand. The timetables that came after 1970 were based on the timetable of the year before. Only a
few adaptations were made. However, since Spoorslag ’70 was a great success and the demand for
transport was growing, the need to use the network as efficiently as possible was increasing. This
lead to the demand for tools that could assist in developing a timetable.

A complicating factor in the planning process is that there are trains with different characteris-
tics that share the same tracks. Sprinter trains accelerate faster than intercity trains, but do stop
at every station, while intercity trains accelerate slower but do not stop everywhere. Furthermore,
there are also freight trains, which are very slow compared to passenger trains. The capacity on
the tracks is used in the best way when all trains with the same characteristics are driven one after
the other, and after that the other trains. A time-space diagram for such a situation is shown
in Figure 1.7(a). The dashed lines correspond to the sprinter trains, the continuous lines to the
intercity trains.

Another method to use the tracks is shown in Figure 1.7(b). In this timetable fast and slow
trains are spread out over the hour, fast and slow trains are alternately driven on the track.
Although there are only eight trains driven in this timetable, compared to 17 in the other, it has
a better service to the customers. They do not need an intercity or sprinter train every three
minutes, but prefer them to be spread out over the hour. Having a sprinter train and an intercity
train leaving every 15 minutes, gives more options to customers to choose from.

Most problems in the planning process do not occur on the tracks between stations. The
problem is mainly around the stations themselves, where train paths might have to cross or when
the capacity limits the number of trains present at a station. Furthermore, the problem planners
face nowadays is to deal with the increasing frequency in which trains drive. Sometimes, trains
drive six times per hour and plans are to increase this on some tracks to eight times. This makes
it very hard to find paths around the stations without getting conflicts.

1.5 Problem description

The increasing requirements that are put to a timetable, like increasing frequencies of train lines
and more connections, increased the need for tools that help planners to develop a timetable.
Therefore, NS started to develop a Decision Support System in 1992 called DONS. This software
package assists in creating a basic hourly pattern for the network. The system will be described
in the next chapter. In DONS, a rail network can be defined, including connections and frequency
constraints. Based on this, a mathematical model is built that is solved by a solver called CADANS.
This model consists of constraints that all have the form

\[ l_{ij} \leq v_j - v_i + T_{pij} \leq u_{ij} \] (1.1)

DONS is the abbreviation for ‘Designer of Network Schedules’. Initially DONS abbreviated to ‘Dienstregeling
Ontwikkeling bij de Nederlandse Spoorwegen’, but this was changed later on.
Here \( i \) and \( j \) are events, \( v_i \) and \( v_j \) denoting the points in time they happen, where \( T \) denotes the cycle time of the timetable and \( 0 \leq v_i, v_j < T \). It means that events \( j \) should happen at least \( l_{ij} \) and at most \( u_{ij} \) minutes after event \( i \). In Chapter 3 the model will be explained including all the constraints.

The Dutch rail timetable of 2007 is based on a timetable generated by this package. The timetables that were operated after 2007 were all based on this schedule. However, demands on the schedule have become larger: one wants to have a higher frequency for many lines and better connections. This often leads to a system in which all feasible solutions are excluded, i.e., it is overdetermined. Hence no feasible timetable exists satisfying all the wishes and constraints needed to operate a safe timetable. In case no feasible timetable is possible, and hence the underlying mathematical model has no solution, CADANS returns a set of constraints that cannot be satisfied simultaneously. This means that the constraints to the timetable are put too tight. They need to be relaxed for a feasible solution to exist.

1.5.1 Conflicts

Originally, when DONS and CADANS were developed, the idea was to build a model in DONS with all the trains that had to be driven and all the connections that had to be realised, and solve it with CADANS in order to get a timetable. Although DONS is perfectly able to build such a model for the complete Dutch rail network, it has not been used in this way to design a timetable. One of the main reasons for this is that it happens very often that conflicts occur in the network. This means that a set of constraints exclude the possibility for a feasible solution to exist. For such a situation, constraints have to be adapted in order to get a feasible schedule.

In practice, the planners start with the international trains which have fixed arrival or departure times at the borders of the country. Once those trains have been planned, the intercity network is added to it. Those are the most important trains for NS. At the one hand, this is because intercity trains need to be fast to provide fast connections between cities. On the other hand, most money is made with the intercity network. Once the intercity network has been fixed, local trains are added. If this does not fit, it is easy to see where problems are in case this does not fit, since the last train that was added has to be involved in the conflict. The timetabling problem is approached in several phases. Although this is an appropriate approach, solving the full network in one go might probably lead to a better timetable, provided the model includes no infeasibility.

In the past, a CADANS solution had to be available in order to see time space diagrams in
DONS. However, nowadays there is another functionality available in DONS in which time space diagrams can be drawn by hand and which shows if conflicts occur between trains, so CADANS is not necessary any longer. It is only used to check a given schedule for feasibility. Therefore CADANS, which is a good solver in the case that a schedule exists, is hardly used any more for the task it was developed for.

In the case that a full network is presented to CADANS to design a timetable, it hardly ever can come up with a solution. Very often there are conflicts occurring. If this is the case, the minimal subset of the constraints that form a conflict are shown to the user. However, it is very hard to determine precisely where the problem arises. Often, the constraints give rise to a complex conflict graph, with many more arcs than nodes. It often consists of a set of trip time or dwell time constraints, together with some safety constraints and some wishes, like frequency or connection constraints. Some of the constraints are more important than others. However, it is very hard to determine which constraint has to be changed and by how much it has to be adapted in order to solve the conflict. Therefore, a tool which automatically resolves the infeasibilities might be useful in order to get good assistance in resolving conflicts and it might lead to having a good timetable.

An example of a conflict that can occur is when a ‘super series’ is defined. Super series are often defined for trains that share a certain part of the network. As an example, consider the 500 series (Rotterdam, Utrecht, Amersfoort, Zwolle, Groningen) and the 1700 series (The Hague, Utrecht, Amersfoort, Deventer, Hengelo, Enschede). Those series share tracks between Gouda and Amersfoort. A picture of the lines is shown in Figure 1.8. They are shown in the time space diagram in Figure 1.5 as well.

![Figure 1.8: Route of the 500 and 1700 series](image)
CHAPTER 1. INTRODUCTION

Suppose both trains have a frequency of two trains per hour, so every 30 minutes a train runs for that series. Further, suppose for simplicity that no other intercity series shares the same track. Then, in order to provide a good service, the intercities should run 15 minutes after each other on the shared part of the network. This will give the best service to the customer. Therefore, they have to be ‘synchronized’ at the start of the shared track, in this case at Gouda. This is done in Figure 1.5 as well, where those intercities synchronize at Gouda.

On the shared part, the two series behave as being one series with a frequency of four trains per hour. However, suppose the 1700 series uses VIRM (see Figure 1.9(a)) and the 500 uses ICM-trains (see Figure 1.9(b)) to operate the line. Those trains have different characteristics, i.e. one might accelerate faster than the other. Therefore, trip times will not be the same on the shared track. This means the one might use less time to travel between two shared stations. However, they should depart and arrive with 15 minutes in between of them. This may easily give rise to conflicts. This means the faster train should have a small delay in order to have the same trip time on the shared tracks.

(a) VIRM: Verlengd InterRegio Materieel ‘Regiorunner’

(b) ICM: Intercity Materieel ‘Koploper’

Figure 1.9: Several types of intercities in use at NS

Another conflict that might occur is when a sprinter train does not fit between two intercity trains on a track, because it consumes more time on the track. Normally a sprinter train leaves shortly after an intercity train, but its average speed is lower than that of an intercity train. Therefore, the next intercity train might get close to the sprinter train running in front of it. This means the driver of the intercity train will see a lot of red signs, stating that he is coming to close. If it is possible to overtake the slower train, it is no problem, but most of the times this is not possible.

A solution to this problem is to delay the intercity slightly, meaning trip times have to be changed. It can either halt for some extra minutes at a station, or it should not drive at full speed, or a combination of both possibilities. If a train has to stop for a red sign, this consumes much more time than when it drives at a lower speed but does not have to stop for some signs. Delaying a train in order to avoid the situation that it comes too close to a slower train running in front of it is called uitbuigen in Dutch.

A sketch of this situation is shown as a time space diagram in Figure 1.10. Here, four intercity trains are running from Zwolle (Zl) to Groningen (Gn), corresponding to the black lines in the figure. Some of them get too close to a sprinter train running in front of it, corresponding to the dashed line. If trains would continue running at full speed, a conflict will occur close to Groningen, as can be seen by the lines crossing each other. Therefore, it is better for the intercity train to
follow the red path, meaning it is delayed a little bit.

Other situations may occur in which there are simply too many trains on the track. In this case it is sometimes necessary to relax frequency constraints. Suppose it is not possible to have a train every 15 minutes (a 15/15 pattern) in some direction. Planners might relax this to have the first train after 10 minutes and the next train 20 minutes after that (a 10/20 pattern).

1.5.2 Problem statement
The previous sections briefly showed that conflicts might occur in a timetabling model. Further, it has shown that there are several ways in which a conflict can be resolved, but it is not always clear what to do to resolve the conflict. Currently planners are working on the timetable for 2018 and they are planning this by hand, since the DONs package cannot produce a timetable that satisfies all requirements and it is hard to determine which constraints need to be relaxed in order to solve the problem. Furthermore, if some problem in the network is solved, another might pop up. Finding an automatic way to find a feasible timetable that might violate the initial constraints is what this research is focussing on. Until now, there is no tool available that generates a timetable in case no feasible timetable exists initially. Therefore, an alternative for, or an extension to CADANS has to be developed to generate a timetable. This leads to the following research question:

*How can a railway timetable be found with minimal deviations from the original constraints when initially no feasible timetable exists?*

In case the initial constraints are such that no feasible schedule exists, they have to be relaxed or adapted in order to obtain a feasible timetable. What ‘minimal deviation’ means will be dealt with later on in this thesis, since that depends on the type of constraints contained in the model. For now, it is important to note that some constraints can be violated, for example constraints concerning transfer times, while safety constraints *always* have to be satisfied. Furthermore, there might be a difference between violating only one constraint by two minutes, or violating two constraints by one minute. Next to that, the current model assumes fixed trip times. Throughout this thesis, this assumption will be valid in some sense as well. For CADANS, they are assumed to be fixed, there is no freedom in there. However, during the algorithms proposed in this thesis, it is allowed to change the trip time a little bit. In Chapter 3 we will come back to this assumption.

The research question can also be reformulated as follows: Given a set of constraints, find an adaptation of the constraints such that a feasible solution to the constraints exists. The relaxations to the constraints should be allowed (no safety constraint can be violated) and the change should be as small as possible.

1.6 Literature review
For a long time, the planning process was done by hand. By now, several papers have been published that describe models to solve the timetabling problem. Many of them study cyclic railway timetables and use models that are based on a Periodic Event Scheduling Problem (PESP) as introduced by [Serafini and Ukovich (1989)](Serafini1989). PESP is described in Section 3.2. Since PESP is NP-complete [Serafini and Ukovich (1989)], several methods have been proposed to solve it.

A PESP-model is developed for Netherlands Railways by Voorhoeve and improved by [Schrijver and Steenbeek (1993)](Schrijver1993), who developed the constraint programming based algorithm CADANS to
solve the model. In general, this algorithm performs well, although instances occur that are hard
to solve by CADANS. Peeters (2003) gives a rather thorough description of all the constraints that
need to be present in a PESP-model in order to obtain a timetable satisfying the safety regulations.

In Liebchen and Möhring (2007), a description is given of what can be included into a PESP-
model and what can not. Topics that can be included are safety, trip time and connection con-
tstraints, topics that are very important for a timetable. Also optional stops and variable trip
times can be included. Topics that can definitely not be included into PESP are calculations of
the routes that trains or passengers should take. Hence, fixed routes of trains are assumed in the
planning. They show that if decisions on network planning, line planning and vehicle scheduling
are to be integrated into PESP, only two features have to be added to PESP, namely a linear
objective function and a symmetry requirement. A linear objective function does not change PESP
much, however, the symmetry requirement does. In a Mixed Integer Programming formulation,
the symmetry requirement is not hard to include and a MIP-solver can deal with this very well.

Another scientist working on models and solvers concerning PESP is Prof. Nachtigall from
Technische Universität Dresden. He has been working on several aspects off line planning and
timetabling. Concerning PESP, he has been working on PESP-models that take an objective
function into account. In Nachtigall and Opitz (2008), a network simplex method is proposed to
minimise a weighted sum of undesirable slack times, e.g., waiting time for passengers. Further,
in Großmann et al. (2012) a polynomial reduction from PESP to SAT\(^{10}\) is proposed. This SAT
model is solved by a state-of-the-art SAT solver and performs very well. Solutions, if they exist,
are found in very short time.

Peeters (2003) and Liebchen (2006) have done research on (minimum) cycle bases, which form
an important part of the constraint graph of any PESP instance. Based on those cycles, the order
of some events can be determined. Further, in Liebchen (2006) a timetable is developed for the
Berlin underground based on a PESP model that has been put into practise.

Many models assume that trip times are fixed. This greatly simplifies the problem. In Kroon
and Peeters (2003), a variable trip time model is introduced. However, in practice it has shown
to be easier to insert some dummy stations in the network and allowing a train to ‘stop’ there for
some period of time if this would be more appropriate, thus increasing the trip time.

The PESP models are not only used to produce a timetable, but also to investigate where
investments in infrastructure are the most profitable, as Odijk (1998) does. His aim is mainly to
find specifications for extensions of infrastructure in or around stations.

At Netherlands Railways, the planning software package DONS is presently in use, with the
solver CADANS developed by Schrijver and Steenbeek (1993). The Dutch timetable of 2007 is
based on the solution computed by this software. Also the rolling stock and crew planning is done
by using Operations Research tools. For this work, the team received the Franz Edelman Award
in 2009 (Kroon et al. (2009)).

### 1.7 Outline of the thesis

In this thesis, an extension to CADANS is proposed. First of all, in Chapter 2 several mathematical
models that are used in this thesis will be described. After that, in Chapter 3 the model that
is used will be explained. Next, the method that the CADANS-solver uses will be explained. In
Chapter 4 an extension to CADANS is proposed to solve some shortcomings. In Chapter 5 the
results of this method are shown. In Chapter 6 conclusions are drawn about the results. In
Chapter 7 the results are discussed and recommendations are done for further research.

---

\(^{10}\)Satisfiability problem. This is a model containing propositional formulas.
Chapter 2

Optimisation techniques

In this Chapter, several optimisation models and techniques will be discussed. First of all, computational complexity will be explained briefly and what that means for solving. After that, Constraint Programming is described. Another technique that is used in this thesis is (Mixed Integer) Linear Programming.

2.1 Computational Complexity

Complexity theory arose around the thirties of the 20th century. At the start of that century, G"odel proved that in every theory of enough complexity, there will be theorems that are impossible to prove. Scientists started looking for the borders, what could be proven and what not. In 1936, the Turing-machine was published, which is a model to show how calculations are done, this formed the start of the growth of complexity theory. Turings model made clear how calculations are done, what to do first and what to do after that. It was also possible to say something about the duration of the calculations, depending on the problem size.

Turings model was the start of algorithms. His model came into use in the second world war, when an electrical machine, the first computer, was build to beat the Enigma code of the Germans.

We all have some idea on the complexity of some calculations. Adding two numbers is ‘easy’, while solving the Travelling Salesman Problem (TSP) is ‘hard’. For the latter, currently no algorithm is available that can guarantee to find the optimal solution (route) within a certain time limit for any problem size (the number of ‘cities’). For small instances, including 5 cities for example, all possible solutions can be checked and the best can be taken. However, for 100 cities, it requires a lot of time to solve it in the same way. To make a clear distinction between ‘easy’ and ‘hard’, one does not look to instances of problems, but to different types of problems, i.e., not to a specific case of a problem, but to the general form of a problem. One wants to be able to solve all instances of a problem (like TSP), and not only a few of them. For this, it is important to know how fast the number of calculations grows that have to be done if the problem size grows. If the growth rate is polynomial in the instance size, the algorithm is said to run in polynomial time and the problem is of class \( P \). Problems for which such a polynomial algorithm exists are ‘easy’ and the class \( P \) is the class of problems that are solvable in polynomial time.

Next to \( P \), there is the class \( NP \). The letters stand for nondeterministic polynomial time. The class basically consists of all problems that, if an instance is provided that has a positive answer, there exists a certificate from which the correctness of the positive answer can be verified in polynomial time. Note that it is not necessary that this certificate can be provided in polynomial time, it just should exist. If the answer to some instance is negative, there does not have to be a certificate that confirms the answer. Consider the following example, taken from [Cook, Cunningham, Pulleyblank, and Schrijver (1998)]: ‘Given an undirected graph \( G \), does \( G \) have a Hamiltonian circuit?’: If it exists, the certificate is some Hamiltonian circuit, although this might be hard to find. If the answer is no, that does not have to be proven.

Obviously, we have \( P \subseteq NP \). Many people believe \( NP \) is larger, although this is still an important open question. Within the class \( NP \), there are problems that are \( NP \)-complete, those

---

1The Clay Mathematics Institute of Cambridge, Massachusetts has established 7 Prize Problems, that are still open important mathematical problems. If one of the problems is solved, the one that does so gets $1 million, see [http://www.claymath.org/millennium-problems](http://www.claymath.org/millennium-problems).
are the hardest problems in \( \mathcal{NP} \). The following definition can be given:

**Definition 2.1 (Cook et al. (1998)).** A problem \( \Pi \) in \( \mathcal{NP} \) is \( \mathcal{NP} \)-complete if every problem in \( \mathcal{NP} \) can be ‘reduced’ to \( \Pi \), in polynomial time.

This means, if a polynomial time algorithm is found for a problem that is \( \mathcal{NP} \)-complete, than all problems in \( \mathcal{NP} \) can be solved in polynomial time, i.e. \( \mathcal{P} = \mathcal{NP} \).

Some instances of problems that are \( \mathcal{NP} \)-complete might be solved very fast, but if the instance size grows, no bound can be given on the running time of the algorithm. Many heuristics are developed for problems in \( \mathcal{NP} \), that ‘solve’ them in reasonable time to a good solution, but not necessarily to an optimal solution.

### 2.2 Constraint programming

Constraint programming is a relatively new method of solving problems. It basically means stating constraints that the solution should satisfy, often without objective function, and then finding a solution to those constraints. A constraint is a logical relation between some variables, thus restricting the possible values for the variables. \( x < y \) does not state which value \( x \) should have, it only states it should be less than \( y \).

A Constraint Satisfaction Problem (CSP) is a problem with:

- A finite set of variables
- A function mapping every variable to a finite domain (values it can take).
- A finite set of constraints

A solution to CSP now is an assignment of a value from its domain to each variable that satisfies all constraints. Note that the constraints can take any form that is wanted.

A method to solve the CSP is by fixing some variable to a value, and propagating this value via the constraints to the other variables thus restricting the domain of those variables. This basically is cleverly searching all possibilities for a feasible solution. Many practical problems are, or can be, solved by constraint programming. It is possible to add an objective function, thus distinguishing several solutions, but it is often hard to find the optimal solution.

### 2.3 Mixed Integer Linear Programming

Linear Programming is a widely used method in optimisation, which is used in this thesis as well in Chapter 3. Many problems can be formulated as a (Mixed Integer) Linear Program. Therefore, many methods have been developed to solve (special kinds of) Linear Programs. Some of them will be explained in this section.

#### 2.3.1 Linear Programming

Linear Programming (LP in short) is a mathematical program where both the objective function and the constraints are linear (in)equalities. In case of a maximisation problem, the standard formulation is the following:

\[
\begin{align*}
\text{maximize} & \quad c^T x \\
\text{such that} & \quad Ax \leq b \\
& \quad x \geq 0
\end{align*}
\]  

(2.1)

where \( x \in \mathbb{R}^n \) is the solution vector and \( b \in \mathbb{R}^m \), \( c \in \mathbb{R}^n \) are known coefficients. \( A \in \mathbb{R}^{m \times n} \) is a matrix with coefficients. \( c^T x \) is called the objective function. Here, there are \( n \) unknowns to be determined, subject to \( m \) constraints. The feasible region \( \mathcal{F} \) is the set of points that satisfy the constraints, i.e.

\[
\mathcal{F} = \{ x \in \mathbb{R}^n : Ax \leq b, x \geq 0 \}
\]

(2.2)

This feasible region is a convex polyhedron. However, there does not necessarily exist a feasible solution. One reason for this is the possibility that the constraints are such that \( \mathcal{F} \) is empty. This might happen if two or more constraints contradict each other. An example is \( x \geq 2 \) and \( x \leq 1 \), which can not be satisfied simultaneously. In this case the LP is called *infeasible*. Next, it might
happen that the polytope is unbounded in the direction of the gradient of the objective function. In this case, there exists no optimal solution. Now, the LP is called unbounded.

If a solution exists and we have a proper objective function (not just finding a feasible solution), the optimal solution is attained at the boundary of the polyhedron. The Simplex Algorithm makes clever use of this property by traversing from one corner of the polyhedron to another.

Example

A small factory produces two types of glasses, for both of them a die is available. The glasses are produced with one machine. With the first die, 100 cases of juice glasses can be produced in six hours. With the second die, the machine can produce 100 cases of cocktail glasses in five hours. The machines are running for 60 hours per week. The produces glasses are stored in a stockroom of 15000 cubic feet. A case of juice glasses requires 10 cubic feet of storage space, while a case of cocktail glasses requires 20 cubic feet.

All glasses are produces for one firm. This firm pays €5.00 for a case of juice glasses and €4.50 for a case of cocktail glasses. Further, the firm does not accept more than 800 cases of juice glasses per week.

The factory owner wants to know how many cases have to be produced of each type of glass such that the turnover is maximized. This can be formulated in an LP-model. Let \( x_1 \) be the number of juice glasses to be produced and \( x_2 \) the number of cocktail glasses (in hundreds of cases per week). Then the objective function to maximize is \( z = 500x_1 + 450x_2 \). The factory runs for 60 hours, giving the constraint \( 6x_1 + 5x_2 \leq 60 \). Next to that, there is only limited storage space available. This gives the constraint \( 10x_1 + 20x_2 \leq 150 \). The customer firm accepts no more that 800 cases of juice glasses per week, giving \( x_1 \leq 8 \). Next, of course the number of glasses produced should be non negative. This gives the following model:

\[
\begin{align*}
\text{maximize} & \quad 500x_1 + 450x_2 \\
\text{such that} & \quad 6x_1 + 5x_2 \leq 60 \\
& \quad 10x_1 + 20x_2 \leq 150 \\
& \quad x_1 \leq 8 \\
& \quad x_1, x_2 \geq 0 \\
\end{align*}
\]  

(2.3)

The feasible region now contains each combination of values for \((x_1, x_2)\) that satisfies the above constraints. Since this is a two dimensional model, this can be shown in a small graph as in Figure 2.1. The shaded region contains is the feasible space. The optimal solution to this problem is

\[
z^* = (x_1^*, x_2^*) = \left( \frac{6}{7}, \frac{2}{7} \right)
\]

(2.4) with an objective value of \( 5142 \frac{6}{7} \).

2.3.2 Integer Linear Programming

In many situations, solutions are wanted to be integer, i.e. \( x \in \mathbb{N}^n \). Solving an LP however might give non integer solutions. Therefore, the decision variables are required to be integer. This leads to the following standard formulation of a maximisation Integer Linear Program (ILP):

\[
\begin{align*}
\text{maximize} & \quad c^T x \\
\text{such that} & \quad Ax \leq b \\
& \quad x \geq 0 \\
& \quad x \in \mathbb{N}^n \\
\end{align*}
\]

(2.5)

where \( x \in \mathbb{N}^n \) is the solution vector, \( b \in \mathbb{R}^m \), \( c \in \mathbb{R}^n \) are known coefficients and \( A \in \mathbb{R}^{m \times n} \) is a matrix with coefficients.

The LP-relaxation is the program in which the integrality condition is skipped. Note that the LP-relaxation will always have at least the same objective value as the ILP, it might be even higher. Solving the LP-relaxation is much easier than solving the ILP. For the LP, the feasible region is a convex polyhedron. However, the ILP consists of a set of discrete points, all the integer points that are contained in the feasible region of the LP-relaxation. The optimal solution of an LP-relaxation might be useless, as it might contain non-integer decision variables. However, using a branch-and-bound algorithm (see Section 2.3.4), it might be used to find the optimal integer solution.
Example continued

As we have seen in the example of the LP formulation, the optimal solution was to produce $6\frac{3}{7}$ cases of juice glasses and $4\frac{2}{7}$ cases of cocktail glasses. Of course, this is not possible, since those are fractional numbers. Now suppose that the factory manager wants to produce multiples of hundreds of cases. This means that the optimal solution according to the LP formulation is no longer feasible. The model now becomes:

$$
\begin{align*}
\text{maximize} & \quad 500x_1 + 450x_2 \\
\text{such that} & \quad 6x_1 + 5x_2 \leq 60 \\
& \quad 10x_1 + 20x_2 \leq 150 \\
& \quad x_1 \leq 8 \\
& \quad x_1, x_2 \in \mathbb{N}_{\geq 0}
\end{align*}
$$

This solution obtained in the previous paragraph now is the solution to the LP-relaxation of the Integer Program. A solution to the IP needs to have integer solutions. So the feasible region now consists of only 54 possible solutions, all the points that are marked by red dots in Figure 2.2.
2.3. MIXED INTEGER LINEAR PROGRAMMING

Now the optimal solution is

\[ z^* = (x_1^*, x_2^*) = (8, 2) \]  (2.7)

with an objective value of 4900. As can be seen, this solution is quite different from the solution for the LP-relaxation. There, the restriction that \( x_1 \leq 8 \) had no effect on the optimal solution. However, if the constraint \( x_1 \leq 8 \) would be left out for the IP, the optimal solution would be to only produce the juice glasses.

2.3.3 Mixed Integer Linear Programming

A combination of the two problems described above is the Mixed Integer Linear Program (MILP). Again, all constraints are linear (in)equalities, but now some of the variables are required to be integer, others are continuous. The standard formulation is as follows:

\[
\begin{align*}
\text{maximize} & \quad c^T_1 x_1 + c^T_2 x_2 \\
\text{such that} & \quad A \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq b \\
& \quad x_1, x_2 \geq 0 \\
& \quad x_1 \in \mathbb{R}^{n_1} \\
& \quad x_2 \in \mathbb{N}^{n_2}
\end{align*}
\]  (2.8)

where \( c_1 \in \mathbb{R}^{n_1}, c_2 \in \mathbb{R}^{n_2}, A \in \mathbb{R}^{m \times (n_1+n_2)} \) and \( b \in \mathbb{R}^m \).

Example continued

Now suppose that the number of cases (in hundreds) of juice glasses is allowed to be relaxed, the owner factory want to find out how that influences the optimal mix of glasses to produce. Than the model becomes the following:

\[
\begin{align*}
\text{maximize} & \quad 500x_1 + 450x_2 \\
\text{such that} & \quad 6x_1 + 5x_2 \leq 60 \\
& \quad 10x_1 + 20x_2 \leq 150 \\
& \quad x_1 \geq 0 \\
& \quad x_2 \in \mathbb{N}_{\geq 0}
\end{align*}
\]  (2.9)

Now the feasible region is no longer restricted to the red dots in Figure 2.2. The feasible now becomes continuous in one of the variables. It now corresponds to the region marked in red in Figure 2.3.

![Figure 2.3: Feasible region example IP](image-url)
Now the optimal solution is
\[ z^* = (x_1^*, x_2^*) = \left( \frac{2}{3}, 4 \right) \]  
(2.10)
with an objective value of \(5133\frac{1}{3}\). As can be seen, this solution is quite different from the solution of the IP, it has moved more towards the solution of the LP-relaxation. Now, the restriction \(x_1 \leq 8\) plays no role any more.

### 2.3.4 Branch and cut

Branch-and-cut is an algorithm that implicitly enumerates all possible integer solutions by using a search tree and adding cuts if fractional solutions are found. Cuts are extra constraints that cut away part of the feasible space for the LP-problem that does not contain any integer solution.

First of all, the LP relaxation is solved using the regular simplex algorithm. In the search tree, each subproblem is called a node and the LP-relaxation of the MIP is the root node for the search tree. If the solution to the LP relaxation contains fractional variables, a cutting plane algorithm tries to add cuts that cut away non-integer solutions from the search space. After that, the branch and bound procedure is started. This algorithm creates two or more sub problems. Suppose for the optimal solution of the LP-relaxation we obtain \(x_k = q\), where \(q \in \mathbb{R} \setminus \mathbb{Z}\). Then two sub problems might be created, one getting the extra constraint \(x_k \leq \lfloor q \rfloor\), the other one getting the constraint \(x_k \geq \lceil q \rceil\). This cuts away the fractional solutions with \(x_k = q\). Note that the objective value of a sub problem will always be smaller than that of the parent problem, since part of the feasible space is cut away, extra restrictions are added. The LP relaxation now forms an upper bound for the objective function and the integer solution (if found) forms a lower bound for the objective function.

After the new nodes are made, the LP-relaxation of those nodes are solved using the Simplex algorithm, and the process starts all over again. In this way, during the search process, hopefully integer solutions will be found. The node that gives the best integer solution is called the ‘incumbent node’. Finding integer solutions provides bounds on the search process, since worse solutions than the incumbent can be rejected. If for some branch it is found that the best possible objective value is lower than that of the incumbent, this branch can be pruned, it can be rejected. No better solutions will be found here. Using the search algorithm, lower bounds can be calculated for the objective value as well. This closes the gap between the best found and the lower bound, leading in the end to the optimal solution.

For branching, it is very important which fractional variable is branched on. There are several strategies implemented in the various MIP solvers, like branching on the nodes that have the least or the most number of fractional variables, branching on the node having the best potential of finding a better solution (strong branching) or branching on the variable that in the past gave the best improvement in the objective function value (pseudo cost branching). For most solvers, it is possible to assign a branching priority to variables, stating which variable to branch on first. This can be done in order to provide more knowledge about the problem to the solver, this speeding up the solving process.

### Branch and bound for IP-example

In Section 2.3.2, an optimal solution was stated for the IP-problem. If some automated tool is used, this solution is found most of the times by a branch and bound algorithm. For this small instance, the branch and bound tree that might be used can be shown. The first problem (a node in the search tree) to start with is the LP-relaxation. If this contains fractional values, restrictions are added. If \(x_1\) has the highest priority for branching, two subproblems can be created. The first gets the constraint that \(x_1 \leq 6\), the second gets \(x_1 \geq 7\). This procedure is repeated for each fractional variable until a feasible integer solution is found. A search tree is shown in Figure 2.4.
Figure 2.4: Branch and bound tree for the example problem
Chapter 3

The model and the current solver

In this chapter, the model that is in use in the planning software will be described. Further, the solver, CADANS, is described that is currently available.

3.1 Introduction

Many railway operators use cyclic timetables and so does NS. This basically means that the timetable repeats itself every hour. So, for example, if a train leaves Utrecht for Amersfoort at 12.37, another train will leave at 13.37. There are variations on this, since in peak hours more trains might be driving than in off peak hours, especially when compared to the beginning or end of a day. However, this hourly pattern forms the basis for the timetable and sets the structure for the whole day. This means that if a timetable is to be designed, an hourly pattern is sufficient for the structure of the timetable.

As mentioned in the previous chapter, NS uses the planning software DONS. This is a software package they started to develop at the beginning of the nineties together with ProRail. Currently, maintenance is done together with ORTEC B.V.

The system consists of two main parts, a ‘definition’ part and a ‘solving’ part. In fact, DONS is the ‘definition’ part, the Graphical User Interface, and it can call the ‘solving’ part CADANS. In the Graphical User Interface, a rail network can be defined with the infrastructure. Given the infrastructure, a train line can be defined on the network. Stops need to be defined and a type of material which will serve the line. For the rolling stock material, characteristics can be given such that trip times can be calculated between train path points. Given the lines, certain wishes can be defined, like connections between trains or spreading some lines out over the hour on a track (synchronizing trains). We will return to this type of requirement later in this chapter. Hence, in DONS a full network can be defined that NS wants to operate. Given all those requirements, DONS can generate a PESP model that, if it is solved by CADANS (the second part of the software), produces a feasible timetable.
and consists of a few modules, as is shown in Figure 3.1. A description of CADANS and its functionalities is given in Schrijver and Steenbeek (1993) and will be described in Section 3.4. Next to the CADANS-module, there is a STATIONS module. Given the solution CADANS has found, the STATIONS module can do the planning at the stations to find out whether there is enough capacity at the platforms. The STATIONS module is not described here since it is not relevant for this thesis.

3.2 The current model

This section is largely based on Peeters (2003). For more information on the model and its properties, see this thesis.

The model that is in use at NS consists of a set of events and a set of constraints. In the rail network, nodes are defined, which are places where conflicts can occur. Those are called train-path points. A train leaving or arriving at such a point is seen as an event for which a moment in time has to be found. First of all, all stations are seen as train-path points. A sketch of the train path-point Utrecht Central Station (as it will be within a few years) is shown in Figure 3.2(a). Everything that is included in the train-path point is within the black frame. Note that this includes all the switches around the station. At both sides of the station, several switches are shown (the points where lines cross). The points at the edges of this area, where the trains enter and leave, are called in-out points, they are marked with red dots.

![Figure 3.2: Two train path points](www.sporenplan.nl)

For small stations, like Krabbendijke, which is shown in Figure 3.2(b), this situation is much easier since they do have only few or no switches at all around them and have only few in-out points. In the model, the stations are seen as ‘black boxes’, what happens within them is not integrated into the timetabling model. The platform assignment is done by the ‘stations’ module. However, some safety issues (if they are now beforehand), are incorporated into the timetabling model. This includes crossing train paths at the stations.

---

1. Calendar Timetabling Algebric Timetable Algorithm for Dutch Railways.
2. Dutch: de wisselstraten
Next to the stations, places where tracks merge or split and bridges are train-path points as well since conflicts can occur here. Bridges might be opened during some times of the day which has to be incorporated into the timetable. Further, at a splitting of tracks, train can not be there at the same time.

Each train passing, entering or leaving a train path point is an event for which a moment in time is to be determined that the event happens. So finding a timetable means finding a point in time for all the events that happen. Those events are related in time to each other by constraints, stating something about the time differences between two events.

The model that DONS uses is a Periodic Event Scheduling Problem (PESP), as introduced by [Serafini and Ukovich] (1989). In order to introduce the definition of PESP, a periodic event and a schedule have to be defined:

**Definition 3.1** (Periodic event, [Serafini and Ukovich] (1989)). A periodic event is a countably infinite set of events \( e(p) \), indexed by \( p \in \mathbb{Z} \), with occurrence times \( t(e(p)) \in \mathbb{R} \) such that for each \( p \), \( t(e(p)) - t(e(p - 1)) = T \). \( T \) is referred to as the period and \( e(p) \) as the \( p \)-th occurrence of the periodic event \( e \). We also define \( t(e) := t(e(p)) \mod T \).

Here \( e(p) \) is an equivalence class of \( \mathbb{R}/T \). Later we come back on this in the definition of a clock time. Further, we need to define when an event is called scheduled

**Definition 3.2** (Scheduled event, [Serafini and Ukovich] (1989)). A periodic event \( e \) is said to be scheduled if an element \( \tau(e) \in \Phi := \mathbb{R}/T \) is associated to it, such that \( \tau(e) = \Pi \cdot t(e(p)) \), for all \( p \), where \( \Pi \) is the canonical projection from \( \mathbb{R} \) to the abelian group \( \Phi \).

Now the definition of PESP can be given as follows:

**Definition 3.3** (PESP, [Serafini and Ukovich] (1989)). The Periodic Event Scheduling Problem is the following one. Given

- a finite set \( A \) of \( n \) periodic events with a common period \( T \);
- a finite set \( A \) of span constraints;

find a schedule \( \tau(e) \in \Phi \), for all \( e \in N \), satisfying all span constraints.

Each span constraint can be written as follows:

\[
(v_j - v_i) \mod T \in [l_{ij}, u_{ij}] \quad \text{for all } (i, j) \in A,
\]

where \( v_i, v_j \) are the moments in time that events \( i \) and \( j \) take place.

In Dutch railway planning practice, \( T = 60 \), meaning that planning is done in full hours, with a time grid in minutes. However, recent experiences show that it might be better to use a finer time grid, like a grid of 30 seconds or even tenths of minutes. This gives a much finer time grid and increases the flexibility in the planning process. Furthermore, a full minute is quite a lot in a busy schedule as the Dutch rail network has. Since the planning software DONS uses full minutes and the current literature also deals with full minutes, this report does the same. However, it is recommended to test the algorithms on instances that deal with a finer time grid. Note that the definition of PESP allows for real times for the events, they are not required to be integer.

Definition 3.3 shows that PESP is a feasibility problem and does not look for an optimal solution, in whatever way optimality will be defined. [Serafini and Ukovich] (1989) show that PESP is NP-complete by a polynomial transformation from the Hamiltonian Circuit Problem.

In order to avoid the word ‘modulo’ in an equation, (3.1) is rewritten as

\[
v_j - v_i \in [l_{ij}, u_{ij}] \mod T \quad \forall (i, j) \in A
\]

To avoid the modulo operator at all, a variable \( p_{ij} \in \mathbb{Z} \) is introduced to denote the time shift from one period to the other:

\[
l_{ij} \leq v_j - v_i + Tp_{ij} \leq u_{ij} \quad \forall (i, j) \in A, \ p_{ij} \in \mathbb{Z}
\]

There are several types of constraints that a timetable should satisfy, they all can be written as (3.3) so they fit into the PESP-model. The constraints that play a role in the planning process will be described. In this description, the departure of train \( t \) from node \( n \) is denoted by \( d^n_t \), and the
CHAPTER 3. THE MODEL AND THE CURRENT SOLVER

arrival of train $t$ at node $n$ by $a_n^t$. Finding a timetable means finding values $d_n^t \in \{0, 1, \ldots, T - 1\}$ and $a_n^t \in \{0, 1, \ldots, T - 1\}$ for all nodes and all trains.

In order to make notation clear and unambiguous, the elements of $\mathbb{R}/T$ are called real clock times. $\mathbb{R}/T$ is the quotient group consisting of the equivalence classes under the relation $T$, which is a relation on $\mathbb{R}^2$ such that $aTb$ if $a = b \mod T$. This means that $b - a$ is an integer multiple of $T$. If $a \in \mathbb{R}$, then there is exactly one $k \in \mathbb{Z}$ such that $a' = a + kT \in [0, T)$. The same can be defined for integer numbers: an integer clock time can be defined as an element from $\mathbb{Z}/T$.

Note that this equivalence relation is reflexive, symmetric and transitive. In this equivalence class, we have the normal relations of addition and subtraction. However, to make the inequality relations clear, define the following for a clock time:

\[
\begin{align*}
    a > 0 & \iff 0 < a' < T/2 \\
    a > b & \iff a - b > 0 \\
    a < b & \iff b > a \\
    a \geq 0 & \iff a = 0 \text{ or } a > 0 \\
    a \leq 0 & \iff a = 0 \text{ or } a < 0
\end{align*}
\]

Those definitions give an ordering of the elements in $\mathbb{R}/T$. Suppose we have $v_1 = 10$ and $v_2 = 50$, where $T = 60$. One would state that $v_1 < v_2$. Using (3.4c), this is equivalent to $v_2 > v_1$. This is equivalent to (using (3.4b)) $v_2 - v_1 > 0$, which is equivalent to (using (3.4c)) $0 < (v_2 - v_1)' < 30$. This would mean that $0 < 40 < 30$ in $\mathbb{R}$, which is not true. Therefore, here in $\mathbb{R}/T$ with the defined inequality relations it holds that $50 < 10$.

In the inequalities as they are defined for $\mathbb{R}$, we can either have $a < b$, $a > b$ or $a = b$. However, now the extra possibility is having $a = T/2$. Suppose there are two trains departing at clock times $v_1$ and $v_2$, that do not depart at the same time. It might happen that train 1 departs before train 2 ($v_1 < v_2$), or the other way around ($v_1 > v_2$). It is also possible that trains depart $T/2$ minutes apart from each other, so $v_1 = v_2 + 30$. Here it is not possible to say whether $v_1 < v_2$ or $v_1 > v_2$, even modulo $T$.

All constraints are written as $v_2 - v_1 \in [l, u]_T$. This means that $v_2 - v_1 \geq l \mod T$ and $v_2 - v_1 \leq u \mod T$. Event 2 should happen at least $l$ and at most $u$ minutes after event 1. This shows that the order of the events is really important.

**Trip times** In the current model that the DONS system is using, it is assumed that trip times are known and fixed beforehand. So given a departure event, the arrival event can be calculated by adding the trip time to the departure event. Doing this gives a huge reduction of the number of nodes in the graph. Basically the number of nodes is halved, giving a smaller problem to solve. In section 3.3.1 after the problem has been described further, it will be shown how the problem can be reduced.

Assuming fixed trip times does give only limited flexibility for the solving procedure. At the one hand this might speed up the solving process, but it might also lead to an infeasibility. Furthermore, some safety constraints depend on the trip times, so assuming flexible trip times causes a difficulty here. More on this further on in this chapter. This problem can be overcome by considering the worst case, but especially for single track parts this might yield an infeasibility because safety constraints are put too tight. Kroon and Peeters (2003) show a method by which it is possible to incorporate flexible trip times into the PESP model and what extra constraints have to be incorporated. One of the solutions they mention is to incorporate dummy stations and give the trains an optional stop here, thus introducing flexibility for the trip times and this is the method that is incorporated in the DONS system as well.

In the derivation of the constraints below, all constraints will be stated in terms of departure events. In the DONS system, a node is a departure event. All nodes are stored in a separate file, giving information about that specific node. Since the node corresponds to a departure event, it also corresponds to a trip and an arrival. Therefore the trip time is stored and the stations between which the train drives.

**Conflicting points** In the rail network, there are a lot of points where conflicts can occur. It is easy to see that conflicts can occur on points which are used for both directions. Examples of such points are stations or single track lines. The two regions at both sides of the point are different. Therefore, we can distinguish between entering and leaving such a conflict point. In practice, a
conflicting point is always a point in the network for which a point in time is to be determined, a train-path point. So we can see trains that enter such a point as incoming trains. Hence, those points are called in-out points.

There should be some time between a train entering and a next train leaving (in-out relation), or the other way around (out-in relation). The minimum time that should be respected between those events are called peak times. It is also possible that trains use a point in the same direction. Also in such a case, a minimum time is needed. This is called a follow-up time. This time might be different if the trains stop or not. So for each pair of trains using a point on the network, a peak- or follow-up time needs to be defined.

**Free track** Places where tracks do not merge and trains just drive are called ‘free tracks’. Here trains can drive after each other in the same direction, or drive in the opposite direction. In case trains drive in the same direction and their speed differs, the fast train can come to close to the slower train, making the fast train come to close to the slower train. Since we do not want trains to see red signs (later more in this), we want to avoid such situations. Further, on single track only one train can use the track at the same time if they travel in opposite direction.

### 3.2.1 Trip time and dwell time constraints

Trains need a certain amount of time to travel from one node to the other. The time needed to do this with the maximum allowed velocity is called the technically minimum trip time. To add some buffer time and avoid small delays, this is increased with a certain percentage, traditionally $5\%-7\%$.

Further, DONS uses only full minutes, so it is rounded to an integer number. It is not rounded to the nearest integer, but it is done in such a way that it leads to a more robust timetable. For the first part of a train line, the focus is put on rounding down, to limit the buffer that is added here. Buffer is mostly needed later on the trip, to avoid getting delays. Hence, later on in the trip the focus on put merely on rounding up. In the DONS system, a small algorithm is implemented that distributes the buffer along the trip. Once a timetable is found, there are evaluation tools that change the timetable to a timetable that is more robust as well.

Consider the 1900 train, travelling between The Hague Central (Gvc) and Venlo (Vl). This train needs 13 minutes to travel from Rotterdam (Rtd) to Dordrecht (Ddr). That means that

$$a_{Ddr}^{1900} - d_{Rtd}^{1900} = 13$$

(3.5)

To write this as a PESP constraint, we rewrite it as

$$13 \leq a_{Ddr}^{1900} - d_{Rtd}^{1900} + Tp \leq 13 \iff a_{Ddr}^{1900} - d_{Rtd}^{1900} \in [13, 13]_T$$

(3.6)

In Dordrecht, the train waits for one or two minute before departing for Breda (Bd). This can be written as

$$d_{Ddr}^{1900} - a_{Ddr}^{1900} \in [1, 2]_T$$

(3.7)

Those two constraints can be combined to the following arrival time independent constraints

$$d_{Ddr}^{1900} - d_{Rtd}^{1900} \in [14, 15]_T$$

(3.8)

### 3.2.2 Frequency and synchronisation constraints

When a line is defined, a frequency has to be provided. This represents the number of times per hour a train leaves to serve this line. So if the frequency is two, the train will be running two times per hour. In this case, it is not wanted to have them leave almost at the same time, but in order to provide a good service, spread them over the hour. That would mean that there should be 30 minutes in between them.

Sometimes two or more lines share a certain part of their route. For example, as mentioned in Chapter 2 the 500 series (Rotterdam to Groningen) and 1700 series (The Hague to Enschede), share a part of their route between Gouda (Gd) and Amersfoort (Amf), see Figure 1.8. Both trains have a frequency of two. In this case, we want those lines to ‘behave’ as one single line on the shared part of their route. Hence, those lines together should have a frequency of four on the shared part. In that case, there departures have to be synchronised, spread out over the cycle time, (at least) at the first station of the shared part, to have them depart there with 15
minutes in between. Because the train from a certain series have a 30/30 pattern, we have the constraint \( d_{Gd}^{300,2} - d_{Gd}^{300,1} \in \{30, 30\} \), as well as \( d_{Gd}^{1700,2} - d_{Gd}^{1700,1} \in \{30, 30\} \). Now we want to have that \( d_{Gd}^{1700,2} - d_{Gd}^{500,1} \in \{15, 15\} \) or \( d_{Gd}^{1700,2} - d_{Gd}^{500,1} \in \{45, 45\} \). This means they are synchronised at Gouda, to drive in a 15/15/15/15 pattern.

In general, suppose there are \( k \) trains travelling in the same direction on a track. In the ideal situation, every \( T/k \) minutes a train should leave. Since this is not always possible, a certain bandwidth \( \delta \) is introduced to allow for some flexibility. Between all trains leaving a certain station, a constraint should be defined to take care of the synchronisation. Let \( d_j \) denote the departure time of the \( j \)-th train. First, consider the departure of trains with respect to the departure of the first train. This means that

\[
\begin{align*}
    d_j - d_1 &\in \left(\frac{(j-1)T}{k} - \delta, \frac{(j-1)T}{k} + \delta\right) \quad \forall j = 2, \ldots, k
\end{align*}
\]  

(3.9)

This means every other train should leave a multiple of \( T/k \) minutes after the first one, plus some flexibility. However, this is all with respect to the first train. If the second and third train leave at

\[
    d_2 = d_1 + (T/k) - \delta, \quad d_3 = d_1 + 2(T/k) + \delta,
\]

the time between those events is

\[
    d_3 - d_2 = d_3 + 2(T/k) + \delta - [d_1 + (T/k) - \delta] = (T/k) + 2\delta.
\]

This means that twice the maximum allowed flexibility is used, while it is desired to have

\[
    d_3 - d_2 \in [T/k - \delta, T/k + \delta]
\]

So the constraints

\[
    d_j - d_i \in \left(\frac{(j-i)T}{k} - \delta, \frac{(j-i)T}{k} + \delta\right) \quad \forall i < j, \quad i, j = 1, \ldots, k
\]  

(3.10)

should be added. This means a constraint is defined between each pair of trains leaving a node, leading to \( k \cdot (k - 1)/2 \) constraints.

However, note that the order of the trains is fixed using those constraints. In this case, we have \( d_i < d_j \) for all \( i < j \), meaning that train \( i \) leaves before train \( j \). So it is better to rewrite the constraints to \( or \)-constraints, thus allowing for more flexibility:

\[
    d_j - d_i \in \bigcup_{n=1}^{k-1} \left[\frac{nT}{k} - \delta, \frac{nT}{k} + \delta\right] \quad \forall i < j, \quad i, j = 1, \ldots, k
\]  

(3.11)

In this formulation, trains leaving at the same moment is prevented and all trains will fall within the desired pattern.

### 3.2.3 Fixed arrival or departure times

International trains arrive at the borders of the country at a fixed time. Planners of the Dutch railways can not change this, those times are internationally arranged. Suppose the 9300 Thalys arrives at the border at node \( 23 \) and \( 27 \). To this end the constraint \( a_{300}^{0} \in \left[23, 27\right] \) is needed. However, this is not a periodic constraint. To change it into a PESP constraint, a node ‘ABS’ is introduced by DONS. This results in \( a_{\text{border}}^{9300} - \text{ABS} \in \left[23, 27\right] \). To make it arrival time independent, but state it in terms of departure events, suppose that the Thalys needs \( r \) minutes to travel from the previous train-path point to the border. Then the constraint can be rewritten to \( a_{\text{border}}^{9300} - \text{ABS} \in \left[23 - r, 27 - r\right] \).

Given some schedule, increasing the times for the events by a certain amount of minutes, leads again to a feasible schedule. Therefore, in the end it is possible to subtract the value for ‘ABS’ from all nodes, in order to have the time of ‘ABS’ set to zero, and hence we have \( a_{\text{border}}^{9300} \in \left[23, 27\right] \).
3.2.4 Safety constraints: headway

The braking distance of an intercity train driving at full speed is about 1100 meters. In case of an emergency break, a train can come to a halt in about 600 meters. This means that when a driver observes something happening and needs to perform an emergency brake, he will be too late most of the time, since seeing what will be happening on the track the coming 600 meters is impossible. Therefore, a safety system is in operation on the railroads. In this system, each track section is divided into blocks. In general those blocks are 1279 meters long, which is more than the braking distance of a train. There is only one train allowed on a block at the same time. At the start of the block there is a signal, indicating whether the block is free or not. If the signal is red, the block is occupied by a train. If the signal is yellow, the block is free, but the next signal will be red indicating the next block is occupied. This gives an indication to the driver to slow down in order to be able to stop at the next signal. If the signal is green, the next signal will not be red, so the coming two blocks are free. An illustration of a situation that might happen is shown in Figure 3.3.

![Figure 3.3: The railway safety block system (retrieved from Peeters (2003))](image)

Block 2 is occupied, causing a red light at the beginning of this block and a yellow light for block 1. In practice, the system is more advanced than described here, but that is irrelevant here. In case the block distance is less than the standard length, it can show, in case of a yellow light, a temporary speed limit in order to be able to stop on time. The length of a block is in general somewhere around 1200 meters. Showing not only the place of the train, but also when the blocks are occupied, would show better whether a timetable would be feasible or not and what the flexibility is. This would give a very detailed diagram, but planners choose to use a less detailed diagram, using a minimum headway time, denoted by $h$, that is needed to separate two trains on the same track for a safe operation. In general this is chosen to be three minutes, corresponding to the time it takes a train to cover a block.

Assume two trains $t_1$ and $t_2$ share a certain track and leave from station $s$. There should be at least $h$ minutes between the trains, so train $t_2$ should leave at least $h$ minutes after $t_1$, leading to the constraint

$$d_{s}^{t_2} - d_{s}^{t_1} \in [h, T]$$

However, train $t_1$ should also leave at least $h$ minutes after train $t_2$, leading to

$$d_{s}^{t_1} - d_{s}^{t_2} \in [h, T]$$

Reversing the last constraint leads to

$$d_{s}^{t_2} - d_{s}^{t_1} \in [-T, -h] = [0, T - h]$$

Combining the two constraints that hold leads to the headway constraint:

$$d_{s}^{t_2} - d_{s}^{t_1} \in [h, T - h]$$

A train may not leave $h$ minutes before or after another train has left for the same track.

It might be possible that trains have a different trip time. Then they can leave some point with $h$ minutes in between, but arrive with less than $h$ in between. To prevent this, it is possible to state the same headway constraint for the arrival events as well.
Suppose that two trains do not have the same travel time. Then they can satisfy the headway constraints on departure, but if the faster train leaves exactly \( h \) minutes after the slower train, the safety constraints will not be satisfied upon arrival. Hence, the same constraint should be stated between arrival events of trains arriving at the same node. However, a model in terms of departure events is wanted, this will be shown in the next section.

3.2.5 Safety constraints: incoming-outgoing and collision

In this section, in-out relations are considered. Two different cases will be reviewed. The first is the situation in which two trains share a point in the same direction. In the second case, two trains share a point in the opposite direction and frontal collisions are to be avoided.

**Same direction** Let \( h_{i}^{12} \) be the follow-up time between the train 1 leaving the conflicting point before train 2. Let \( r_i \) \( (i = 1, 2) \) be the trip time of train \( i \) on the free track after the conflicting point. Either train 1 or 2 departs first. In Figure 3.4 a sketch of a time space diagram is shown.

![Figure 3.4: Out-out and in-in relations](image)

Suppose train 1 departs before train 2 from point \( s_1 \), meaning that \( d_2 \geq d_1 \) or \( d_2 - d_1 \geq 0 \). In that case a minimal follow-up time \( h_{oo}^{12} \in [0, T/2] \) is needed. This means that \( d_2 \geq d_1 + h_{oo}^{12} \) or, equivalently, \( d_2 - d_1 \geq h_{oo}^{12} \).

In the same way, assume that train 2 departs first, meaning \( d_2 - d_1 \leq 0 \). In this case one obtains \( d_2 + h_{oo}^{12} \leq d_1 \) or \( d_2 - d_1 \leq -h_{oo}^{12} \). Combining the two relations leads to the out-out constraint

\[
d_2 - d_1 \in [h_{oo}^{12}, T - h_{oo}^{21}]_T
\]

(3.13)

After having left \( s_1 \), the trains arrive in \( s_2 \). Suppose train 1 is the first to enter, so \( a_1 \leq a_2 \). Then a minimum follow up time of \( h_{ii}^{12} \) is needed, meaning that \( a_1 + h_{ii}^{12} \leq a_2 \) should hold. This leads to \( a_2 - a_1 \geq h_{ii}^{12} \).

Now, if train 2 enters first \( (a_2 \leq a_1) \), then \( h_{ii}^{21} \) minutes are needed, leading to the constraint that \( a_2 + h_{ii}^{21} \leq a_1 \), or \( a_2 - a_1 \leq -h_{ii}^{21} \). Since \( a_i = d_i + r_i \) \( (i = 1, 2) \), this leads to the constraints

\[
d_2 - d_1 \geq h_{ii}^{12} + r_1 - r_2
\]

(3.14a)

\[
d_2 - d_1 \leq -h_{ii}^{21} + r_1 - r_2
\]

(3.14b)

or taken together

\[
d_2 - d_1 \in [h_{ii}^{12} + r_1 - r_2, T - h_{ii}^{21} + r_1 - r_2]_T
\]

(3.15)

**Opposite direction** Suppose two trains share point \( s_1 \) in the opposite direction when travelling to and from \( s_1 \). In the case that the trains run between \( s_1 \) and \( s_2 \), this situation is shown in Figure 3.5, although for the constraints it is not necessary they travel between the same stations.

It is important to note that the order of the events is really important here. Suppose that the departing train departs after the incoming train has arrived (in-out relation). Then the train can start driving once the other train is in the station and has passed the conflicting point. So the arrival and departure events can happen simultaneously. However, suppose one wants the train to depart before the incoming train has arrived (out-in relation). Then those events happening at the same time has to be excluded. If it happens the same time, the order of the events is not clear, it
is seen as departing after an arrival. Therefore, if the relation in time is first a departure and after
that an arrival, those events can not happen at the same time.

Suppose that train 1 wants to depart before train 2 arrives, so \( d_1 \leq a_2 \). As stated before, this
can not happen at the same moment, therefore we need \( d_1 < a_2 \). Further, a minimum peak time
between those events of \( p_{io}^{12} \) has to be respected, giving \( a_2 \geq d_1 + p_{io}^{12} \). Together this gives the
relation \( a_2 \geq d_1 + \max\{p_{io}^{12}, 1\} \). Since \( a_2 = d_2 + r_2 \), this leads to:

\[
d_2 - d_1 \geq -r_2 + \max\{p_{io}^{12}, 1\}
\]  (3.16)

If train 2 enters before train 1 leaves (\( a_2 \leq d_1 \)), those events can happen at the same time. A
train can start driving as soon as the incoming train has arrived. Further, a minimum peak time
of \( p_{io}^{21} \) is needed between those events, leading to \( a_2 + p_{io}^{21} \leq d_1 \), or \( d_2 - d_1 \leq -r_2 - p_{io}^{21} \). Together
this gives the constraint

\[
d_2 - d_1 \in [-r_2 + \max\{p_{io}^{12}, 1\}, T - r_2 - p_{io}^{21}]_T
\]  (3.17)

**Avoid frontal collision** Now suppose the trains share the same single track. Then the trains
can not meet each other on the track, only at the station or passing points if that is allowed.

Consider the situation at \( s_2 \). Suppose train 1 enters before train 2 leaves, so \( d_2 \geq a_1 \). Here we
need to have that \( d_2 \geq a_1 + p_{io}^{12} \) to have safety. This leads to the constraint that \( d_2 - d_1 \geq r_1 + p_{io}^{12} \).

If train 2 departs before the arrival of train 1, then equality is forbidden again. Further, there
should be at least \( p_{io}^{21} \) between the events. This leads to \( d_2 - d_1 \leq r_1 - \max\{p_{io}^{21}, 1\} \). Taken together
this gives the constraints

\[
d_2 - d_1 \in [r_1 + p_{io}^{12}, T - r_2 - p_{io}^{21}]_T
\]  (3.18)

Note that this is the same equation as (3.17), only the bounds are interchanged since now \( d_1 \) and
\( d_2 \) are seen from station \( s_1 \).

Now in order to avoid collisions on single track, the in-out constraints are required at both end
points of the single track. So the intersection of the intervals can be taken to state the single track
constraints as one constraint:

\[
[-r_2 + \max\{p_{io}^{12}, 1\}, T - r_2 - p_{io}^{21}]_T \cap [r_1 + p_{io}^{12}, T + r_1 - \max\{p_{io}^{21}, 1\}]_T
= [r_1 + p_{io}^{12}, T - r_2 - p_{io}^{21}]_T
\]  (3.19)

The generation of this type of constraints clearly shows the difficulty in using flexible trip times,
since many safety constraints depend on the trip time. If a model is used that depends on arrival
events as well, it is possible to allow for flexible trip times. However, this increases the problem
very much, so it is questionable whether the benefits of considering arrival events as well are larger
then the benefits gained by merging the graph to a smaller problem. Kroon and Peeters (2003)
have shown what additional constraints are required in order to allow for variability in the trip
times. However, according to Kroon (2015), it was rather hard to implement and the solution that
was taken was to add dummy stations and allow trains to stop there, thus introducing variability
in the trip times.

\( ^3 \)DONS does this in an other way. There it is assumed that \( a_2 > d_1 + p_{io}^{12} \) giving the constraint \( a_2 \geq d_1 + p_{io}^{12} + 1 \)
3.2.6 Safety constraint: overtaking

Taking into account only the headway constraints, does not necessarily lead to a feasible timetable. Suppose train $t_2$ is faster than train $t_1$ and both travel from $s_1$ to $s_2$. If $d_1 = 0$, $d_2 = 3$, $a_2 = 18$ and $a_1 = 21$, we have that $d_2 - d_1 \in [3, 57]$ and $a_2 - a_1 \in [3, 57]$, so the headway constraints are satisfied. However, in this schedule train $t_2$ should have overtaken $t_1$ on its way, which is impossible, unless there are two parallel tracks. But the safety constraints are only required if two trains share the same track. This situation is shown in a time-space diagram in Figure 3.6.

![Time-space diagram](image)

Figure 3.6: Trains overtaking each other (based on Kroon and Peeters (2003))

In this figure, $t_2$ is the faster train and the grey line shows that it overtakes $t_1$. The trip times of train $t_1$ and $t_2$ are respectively denoted by $r_1$ and $r_2$. The figure shows that it is required to have at least $X = r_1 - r_2 + h$ minutes between the trains. The in-in relations are already taken care of, so the headway time $h$ can be left out. In order to properly derive a safety constraint, let the moment in time the train is at the relative position $\lambda$ of the track be denoted by

$$t_\lambda = d_t + r_t \cdot \lambda, \quad \lambda \in (0, 1)$$

Here $d_t$ is the departure time, $r_t$ is the trip time and $\lambda$ is the relative position on the track.

If train 1 and 2 are at the same time at the same point, then there exists some $\lambda^*$ and $n_1, n_2 \in \mathbb{Z}$ such that

$$d_1 + n_1T + r_1\lambda^* = d_2 + n_2T + r_2\lambda^* \quad (3.20)$$

This is equivalent with stating that

$$(d_1 - d_2) + T(n_1 - n_2) = (r_2 - r_1)\lambda^* \quad (3.21)$$

Now suppose that $r_2 > r_1$, then there exists $m \in \mathbb{Z}$ ($m = n_1 - n_2$), such that $0 < d_1 - d_2 + mT < r_2 - r_1$, which is the same as $d_2 - d_1 \in [0, T + r_1 - r_2]T$. If $r_1 > r_2$, then in the same way it results in $d_2 - d_1 \in [r_1 - r_2, T]T$.

Now note that if $r_1 = r_2$ this constraint does not imply any restriction. Further, if $r_1 = r_2 \pm 1$, then it results in either the interval $[0, T - 1]$ or $[1, T]$ and hence no integer clock times are forbidden. Hence, this constraint only is required in the trip times do differ more than one minute. The safety restrictions at the in-out nodes will guarantee safety at those points.

3.2.7 Safety constraints: hindrance

As is mentioned previously, trains pass train-path points. Those are places where conflicts can occur. In the beginning of this chapter, it was mentioned that train path points are seen as black
boxes, what happens within them is unknown. However, to some extent this is not totally true. Train-path points can be divided into the following categories:

1. Connecting/merging tracks with only fly-overs.
2. Crossings without conflicts.
3. Stops on free track.
4. Connecting/merging tracks with at least one hindering.
5. Crossing with conflicts.

The first three types of train-path points do not have any conflicts, those are covered in the previous sections already. The other two will be dealt with in this section.

Connections with at least one hindering

This situation is a place where tracks merge or split and where train paths cross each other. An example is shown in Figure 3.7. Here, the paths of the trains going from Amersfoort to Apeldoorn (the red line) and the trains from Barneveld to Amersfoort cross each other within the train-path point ‘Barneveld aansluiting’. This means the arrival of those trains have to be separated in time in order to avoid a collision. This can not be covered in ‘in-in’ or ‘in-out’ constraints, since those paths do not share the same points where they enter or leave the train-path point.

![Figure 3.7: Crossing trainpaths at Barneveld (Source: www.sporenplan.nl)](https://www.sporenplan.nl)

The route through the train path point shares a certain point and this might lead to a collision if this is planned in the wrong way. One of the most well known Dutch examples of such a point, where it went wrong although that was not due to planning, is around Harmelen, where two train collided on January 8, 1962. This conflict is removed by now and a fly-over is placed, to avoid having such a situation again. The situation as it is now is shown in Figure 3.8. This is a train-path point of type 1.

![Figure 3.8: Fly-over at Harmelen (Source: www.sporenplan.nl)](https://www.sporenplan.nl)

An important property of this type of conflicting points is that they are geographically very small and trains never stop here. This implies that the arrival and the departure of a train from this point happen in the same minute. Note that if two trains share this point and might have a conflict, there are always four in-out points of the train-path point involved in the conflict. Two for the first train, and two others for the second train. This is because the train paths only cross each other within the train-path point, they do not share the same in-out points.

For more information, the interested reader is referred to [https://en.wikipedia.org/wiki/Harmelen_train_disaster](https://en.wikipedia.org/wiki/Harmelen_train_disaster)
Suppose there is a conflict between trains 1 and 2. If train 1 passes the point before train 2, so \( a_2 \geq a_1 \). At least \( p_{12} \) minutes have to be between those trains, so \( a_2 - a_1 \geq p_{12} \). For the other order, train 2 going first, \( p_{21} \) minutes have to be in between. This leads to \( a_2 - a_1 \leq -p_{21} \). Now note that the arrival and departure event at such small points happen in the same minutes. Combining this all leads to the constraint
\[
d_2 - d_1 \in [p_{12}^T, T - p_{21}^T]
\] (3.22)

**Crossing with conflict**

This conflicting situation is generally around stations. Here, several train lines meet each other and their paths might cross. Consider the train coming from Rotterdam going to the Hague Central (Gvc) together with the train coming from The Hague Laan van NOI (Laa) going to Rotterdam. Their paths cross just before The Hague HS. This situation is shown in Figure 3.9 the green line is the path of the train going to The Hague Central Station, the blue line is the one going to Rotterdam. At the marked red point, they have to use the same switch. This has to be avoided in the planning procedure. If not, a feasible timetable might come out, satisfying all in-out relations, but within the stations it is infeasible.

![Figure 3.9: Crossing trainpaths in The Hague (edited picture of www.sporenplan.nl)](www.sporenplan.nl)

If some shared point is used in the same direction and trains do not stop, no extra restrictions are needed, safety is guaranteed by the follow up times at the in-out points. If trains do stop, their stopping times are not known on foremost, and therefore this might give a problem as well. Further, a problem is in the peak times, when some infra point is used in opposite direction.

In case the infrastructure point is used in the same direction, there should be a certain follow up time at the shared point. The out-out constraint that is seen in Section 3.2.5 is the following:
\[
d_2 - d_1 \in [h_{\text{ioi}}^T, T - h_{\text{ioo}}^T]
\] (3.23)

Now the same constraint is to be stated if this situation occurs at a certain infrastructure point, now with the follow up times for this point.

If the point is used in the same direction, the in-out constraints that were stated were the following
\[
d_2 - d_1 \in [\max\{p_{\text{ioi}}^{12}, 1\} - r_2, T - r_2 - p_{\text{ioo}}^{21}]_T
\] (3.24)
\[
d_2 - d_1 \in [r_1 + p_{\text{ioo}}^{21}, T + r_1 - \max\{p_{\text{ioi}}^{21}, 1\}]_T
\] (3.25)

Again, the same constraint is to be stated, but now with the peak times for the shared infrastructure point.

For the in-in situation, the constraint is
\[
d_2 - d_1 \in [h_{\text{ihi}}^{12} + r_1 - r_2, T - h_{\text{ihi}}^{21} + r_1 - r_2]_T
\] (3.26)

Also here the same holds, the same constraint is to be stated.

This section clearly shows it does not give any new type of constraint, it is basically the same as was seen already. For many situation, the peak and follow up times at infrastructure points at the stations are smaller than those at the in-out points. Many hindering constraints therefore do not impose any new restrictions. However, for others the bounds get tighter.
3.2.8 Connection constraints

At larger nodes, like the station of Eindhoven, passengers should have the possibility to take a connecting train to another direction. They need enough time to walk to that train, but do not want to wait too long. Sometimes this also happens when two trains are to be combined into one train when they share a certain track, what used to happen for trains coming from Rotterdam and The Hague, that were combined at Utrecht, to travel to Zwolle.

Consider the 1900 train arriving at Eindhoven (Ehv) travelling to Venlo. At about the same time, the 800 train from Alkmaar travelling to Maastricht should be at Eindhoven, to let passengers from the 1900 take the 800 train towards Maastricht and vice versa. Two minutes are needed for passengers to go from one train to the other. This means that if the 1900 arrives at Eindhoven, the 800 should leave at least 2 minutes later, and at most 5 minutes, in order to have a good connection. This gives the constraint

$$d_{800 \text{Ehv}} - d_{1900 \text{Ehv}} \in [2, 5]T$$

The time needed to go from one train to the other depends on the type of connection. If it is a cross-platform connection, meaning that two trains are at the opposite sides of the platform, passengers only have to cross the platform. If it is not a cross-platform connection, passengers have to go from one platform to another and more time will be needed. The DONS system takes into account this difference by setting the time interval for the connection constraints.

In Section 3.2.2 it was mentioned that lines are often served more than once per hour, most of the times they have a frequency of two. Now suppose a connection is defined between two lines, line $A$ connects to line $B$. Then each train of the feeder line $A$ should connect to a train of the next line $B$, depending on the arrival and departure times at the station they have the connection.

If some train of line $A$ makes a connection to some train of line $B$, it should make a connection to the first train that leaves after the $A$-train has arrived. It is not wanted to state on forehand which train of $A$ should make a connection to which train of $B$. If a train of $A$ arrives, a next train of $B$ should leave with $c$ and $c$ minutes.

This situation is shown in Figure 3.10. Then train $t_{1,1}$ can be connected to train $t_{2,1}$ (and hence $t_{1,2}$ should connect to $t_{2,2}$). This corresponds to the blue arcs. But it is also possible that train $t_{1,1}$ connects to $t_{2,2}$ and $t_{1,2}$ to $t_{2,1}$. This corresponds to the red arcs. We want to be the connection time to be in $[c, \bar{c}]$.

Preferably one does not want to state beforehand which trains should connect to each other. This leads to the constraints

$$d_{t_{2,1}} - a_{t_{1,1}} \in [c, \bar{c}]T \quad \text{and} \quad d_{t_{2,2}} - a_{t_{1,2}} \in [c, \bar{c}]T \quad (3.28a)$$

OR

$$d_{t_{2,2}} - a_{t_{1,1}} \in [c, \bar{c}]T \quad \text{and} \quad d_{t_{2,1}} - a_{t_{1,2}} \in [c, \bar{c}]T \quad (3.28b)$$

Suppose that

$$d_{t_{2,1}} - a_{t_{1,1}} \in [c, \bar{c}]T$$

The trains drive with frequency 2 and since the frequency-window $[s', \bar{s}]$ is symmetric, we have $s' + \bar{s} = T$. Hence, we derive that

$$d_{t_{2,2}} - a_{t_{1,2}} \in [c + s', \bar{c} + \bar{s}]T$$

is satisfied as well.
Using similar arguments, we can derive that

\[ d_{t,1} - a_{t,1} \in [c, \bar{c}]_T \text{ or } d_{t,2} - a_{t,1} \in [c + s', \bar{c} + s']_T \text{ and } (3.31a) \]

\[ d_{t,1} - a_{t,1} \in [c, \bar{c}]_T \text{ or } d_{t,2} - a_{t,1} \in [c + s', \bar{c} + s']_T \text{ and } (3.31b) \]

\[ d_{t,2} - a_{t,1} \in [c, \bar{c}]_T \text{ or } d_{t,2} - a_{t,1} \in [c + s', \bar{c} + s']_T \text{ and } (3.31c) \]

\[ d_{t,2} - a_{t,1} \in [c, \bar{c}]_T \text{ or } d_{t,2} - a_{t,1} \in [c + s', \bar{c} + s']_T \text{ and } (3.31d) \]

This can be written as

\[ d_{t,1} - a_{t,1} \in [c, \bar{c}]_T \cup [c + s', \bar{c} + s']_T \text{ if } (3.32a) \]

and

\[ d_{t,1} - a_{t,1} \in [c, \bar{c}]_T \cup [c + s', \bar{c} + s']_T \text{ if } (3.32b) \]

and

\[ d_{t,2} - a_{t,1} \in [c, \bar{c}]_T \cup [c + s', \bar{c} + s']_T \text{ if } (3.32c) \]

and

\[ d_{t,2} - a_{t,1} \in [c, \bar{c}]_T \cup [c + s', \bar{c} + s']_T \text{ if } (3.32d) \]

This can be rewritten to arrival event independency by using that \( a_{t,2} = d_{t,2} + t_{t,2} \) and substituting this in the above.

Note that several or-constraints on two variables can be written as and-constraints. Suppose we have that

\[ x - y \in [l_1, u_1]_T \text{ and } x - y \in [l_2, u_2]_T \]

where \([l_1, u_1]_T \cap [l_2, u_2]_T = \emptyset\). Then this can be written as

\[ x - y \in [l_1, u_2]_T \cap [l_2, u_1 + T]_T \]

Graphically this can be seen as in Figure 3.11. The original intervals are shown in red, the new intervals are shown in blue.

![Figure 3.11: Changing and/or constraints](image)

In general, suppose we have

\[ x - y \in [l_1, u_1]_T \cup [l_2, u_2]_T \cup \ldots \cup [l_n, u_n]_T \]

where \(0 \leq l_1 \leq u_1 < l_2 \leq u_2 < \ldots < l_n \leq u_n < l_1 + T\), then this can be written as

\[ x - y \in [l_1, u_1]_T \cap [l_2, u_2 + T]_T \cap [l_3, u_2 + T]_T \cap \ldots \cap [l_n, u_n + T]_T \]

In Peeters (2003) a full section is devoted to flexible connections. Also, how to model connections like one-to-many are explained.
When a train arrives at its final destination, it is often used to serve the line in the backward direction. It needs some time to turn around. Essentially this is a constraint of the same type as the connection constraints. The train in the incoming direction should have a connection to the train in the other direction. However, in general the allowed time window is different from the time window allowed for passenger connections.

In order to make it clear how the constraints arise in a practical situation, two examples are included in Appendices D and E. The first one considers a part of the Dutch network for which a feasible solution exists. The second example considers an artificial example, based on the Dutch infrastructure and that considers single track. It clearly shows the difficulties and the extra constraints that come into play when considering single track.

3.3 Properties of the model

3.3.1 Constraint graph

The set of events and constraints can be represented by a graph \( G = (V,A) \). The events correspond to nodes \( v \in V \) and the constraints correspond to arcs \( a \in A \) defined between two nodes. A node is in fact a triple of a train, a node in the network, and an arrival or departure. Each constraint can be represented by an arc. A constraint like

\[
v_j - v_i \in [l, u]_T
\]

(3.37)

corresponds to an arc from node \( i \) to node \( j \), with a process time in the interval \([l, u]_T\). Graphically this can be seen as in Figure 3.12. Without loss of generality it can be assumed that \( 0 \leq l < T \)

![Figure 3.12: A constraint in the graph](image)

and \( 0 \leq u - l \leq T - 2 \). The main problem now is to find values for the nodes \( v_i \in \mathbb{Z}/T, i \in V \), satisfying all the constraints (arcs) and so producing a feasible timetable.

3.3.2 Reducing the problem

As mentioned in Section 3.2.1 knowing the departure times and given the trip times, it is possible to reduce the model to a model with only departure times. In the description of the generation of the constraints, all constraints were stated in terms of departure events already. So the model that DONS makes, considers only departure events. The trip times are stored in a separate file, which is used to build the timetable including the arrival events, but the solving is done only on the departure events.

Consider the graph in Figure 3.13 showing the trips of a train on a network, serving a line in both directions.

![Figure 3.13: Trip and dwell arcs](image)

Given the trip time, the arcs going from a departure node to another departure node can be merged into one trip-dwell arc, resulting in the graph shown in Figure 3.14. Doing this merging of arcs gives a huge reduction of the number of nodes in the graph. If there is an arc from \( v_1 \) to \( v_2 \)
and an arc from \( v_2 \) to \( v_3 \), with intervals \([l_1, u_1]\) and \([l_2, u_2]\) respectively. Then those arcs can be merged to an arc from \( v_1 \) to \( v_2 \) with interval \([l_1 + l_2, u_1 + u_2]\).

This merging can also be done for other constraints which allow for no flexibility. So consider a frequency constraint, requiring the difference between two events to be in \([30, 30]\), this arc could be merged with others as well. In fact, arcs allowing for flexibility in the process time can be merged with others as well. This is done by CADANS as well, if a node has a degree of at most two. This is shown in section 3.4.1.

There are more possibilities to reduce the problem size. Consider two trains of the same line with a frequency of two, travelling from node \( A \) to node \( B \). Since they leave from the same station, we have a headway constraint between those events. Further, since they serve the same line having a frequency of two, we have a frequency constraint between them. This is shown in Figure 3.15, where the headway time is taken to be \( h = 3 \) and the bandwidth for the frequency constraints is taken to be \( \delta = 1 \).

This figure clearly shows that the headway constraint is redundant, it is included in the frequency constraint, so it can be reduced from the graph. Sometimes the situation occurs in which two intervals are not included in each other, but partly have a common interval. For example, if \( v_j - v_i \in [3, 25] \) and \( v_j - v_i \in [16, 44] \). This leads to \( v_j - v_i \in [16, 25] \), so two arcs can be merged into one new arc. In the description of ConFlex (the module of CADANS that finds a timetable), a preprocessing step is described which does a lot of merging of arcs to simplify the problem, see section 3.4.1.

3.3.3 If the p-values are given

As mentioned before, if the modulo operator is removed, each constraint contains a \( p \)-variable, denoting the shift from one hour to the other. Schrijver (2008) explains how to find a solution easily if the \( p \)-values are given. Suppose \( \pi : V \to \mathbb{Z} \) is a solution, then there exists a function \( p : A \to \mathbb{Z} \) such that

\[
\pi_j - \pi_i \in [l_{ij} + T p_{ij}, u_{ij} + T p_{ij}], \quad \forall a = (i, j) \in A
\]  

(3.38)

This can be approached from the other side, by finding a function \( p : A \to \mathbb{Z} \) such that there exists a function \( \pi : V \to \mathbb{Z} \) satisfying (3.38). Given \( p : A \to \mathbb{Z} \), it is easy to check whether \( \pi : V \to \mathbb{Z} \) exists, by using a variant of the Bellman-Ford shortest path algorithm:

**Algorithm 3.4 (Schrijver 2008)**. Set \( \pi_i = 0 \) for all \( i \in V \).

For all \( a = (i, j) \in A \) with \( \pi_j - \pi_i \notin [l_{ij} + T p_{ij}, u_{ij} + T p_{ij}] \), do:

- If \( \pi_j - \pi_i < l_{ij} + T p_{ij} \), i.e. if \( \pi_j < \pi_i + l_{ij} + T p_{ij} \), increase \( \pi_j \) to \( \pi_j := \pi_i + l_{ij} + T p_{ij} \).
- If \( \pi_j - \pi_i > u_{ij} + T p_{ij} \), i.e. if \( \pi_i < \pi_j - u_{ij} - T p_{ij} \), increase \( \pi_i \) to \( \pi_i := \pi_j - u_{ij} - T p_{ij} \).
Repeat the procedure \(|V|\) times.

If \(\pi: V \rightarrow \mathbb{Z}\) satisfies all requirements, a solution for (3.38) is found. If not, it can be proven that no solution exists for this \(p: A \rightarrow \mathbb{Z}\), so another \(p: A \rightarrow \mathbb{Z}\) should be found. However, finding the right \(p\)-values is difficult. Therefore, the number of possible \(p\)-values should be reduced as much as possible. Schrijver and Steenbeek (1993) show different methods how this can be done and also mention how effective the reduction is. As an example they use a problem with \(n = 537\) nodes and \(m = 4681\) arcs. For arcs where \(l = u\), it is necessary to have \(p = 0\). This happens for example with trip times or frequency constraints. In those cases, also a node can be removed since it is implied by the other node from the arc. This leads to \(n = 235\) nodes, \(m = 3732\) constraints and 3732 \(p\)-variables. Merging parallel arcs leads to \(n = 235\) and \(m = 1274\). Finally, they show that if a spanning tree is found, we can choose \(p = 0\) on the arcs of the tree. This gives a reduction to 1040 \(p\)-variables. This shows that reducing the graph where possible gives a huge reduction in the size of the problem.

### 3.3.4 Safety clique

Next to the train cycles, safety cliques are an important ingredient of the constraint graph. Consider a point in the network, where trains depart. Between each pair of trains, a headway-constraint is defined in order to guarantee a safe timetable. This means that the subgraph containing all nodes in the graph corresponding to a departure from that point in the network, forms a clique. This is often called a safety-clique since it represents constraints that have to do with safety. Since there are very many of such points, there will be a lot of safety cliques in the constraint graph.

### 3.3.5 Cycles

The constraint graph of a railway PESP model is very dense, there are far more constraints than nodes. Therefore, many (directed) cycles can be found in the constraint graph. Consider for example two trains \(t_1\) and \(t_2\) sharing the same track between two train path points. In order to guarantee safety, they are separated on the track by two headway constraints. In the constraint graph \(G\), this corresponds to the left graph of Figure 3.16, where the lower- and upper bounds for the trip times are denoted by \(r\) and \(T\). In the right graph, all arcs are turned into the same direction by reversing some of them. Simply reversing the arc between the lower two nodes would give the interval \([-r_{t_2}, -r_{t_1}]\). However, the intervals are periodic, so it can be shifted by one or more cycle periods in order to obtain an interval with positive bounds. Under the assumption that \(r_{t_2} < T\), this would become \([T - r_{t_2}, T - r_{t_1}]\). However, there are constraints in practice that have an upper bound that is larger than \(T\). This would mean that it is necessary to add an extra cycle period to have an interval with positive bounds.

![Figure 3.16: Small cycle for two trains on a track](image)

If we traverse the cycle in the graph, starting from \(A\), certain process times are found. However, when ending in \(A\) again, the process times should add up to a multiple of \(T\), otherwise we would get a conflict in \(A\), then one event should happen at two different points in time. For the left graph in Figure 3.16, the sum of the process times would lead to \(r_{AB} + r_{BD} - r_{CD} - r_{AC} = q_C \cdot T\) with \(q_C \in \mathbb{Z}\).

A cycle can be identified by its nodes and by its edges. If a cycle \(C \in G\) is considered, we mean by \(C\) the set of edges of the cycle, so \(C \subseteq A\). Further, when we write \(C = (1, 2, \ldots, k, 1)\), a cycle going from node 1 to \(k\) and back to 1, we mean the set of edges between those nodes. The set of forward arcs is denoted by \(C^+\) and the set of backward arcs by \(C^-\). Given a cycle \(C \in G\), we have the following theorem by Odijk (1996) (as cited in Peeters (2003), who gives the proof as well.).
Theorem 3.5 (Odijk (1996)). A PESP instance defined by $G = (V, A)$ and $T$ is feasible if and only if there exists an integer vector $p$ such that, for each cycle $C \in G$,

$$a_C \leq \sum_{(i,j)\in C^+} p_{ij} - \sum_{(i,j)\in C^-} p_{ij} \leq b_C \quad (3.39)$$

where $a_C$ and $b_C$ are defined by

$$a_C = \left[\frac{1}{T} \left( \sum_{(i,j)\in C^+} l_{ij} - \sum_{(i,j)\in C^-} u_{ij} \right) \right] \quad (3.40)$$

$$b_C = \left[\frac{1}{T} \left( \sum_{(i,j)\in C^+} u_{ij} - \sum_{(i,j)\in C^-} l_{ij} \right) \right] \quad (3.41)$$

This theorem shows how many multiples of $T$ are possible in a cycle. In this theorem, the backward arcs are simply reversed to arcs with an interval $[-u_{ij}, -l_{ij}]$, without shifting the interval such that $0 \leq -u_{ij} \leq T - 1$, since this would make the calculations more complicated. However, since the intervals are periodic, it is possible to shift the interval such that the above holds. Consider a constraint of the form

$$l_{ij} \leq v_j - v_i + Tp_{ij} \leq u_{ij}$$

Multiplying the equation by $-1$ leads to

$$-u_{ij} \leq v_i - v_j - Tp_{ij} \leq -l_{ij}$$

$p_{ij}$ denotes the number of multiples of $T$ being added, so we can also add an integer multiple of $T$ to each term of the equation, leading to

$$T\overline{p} - u_{ij} \leq v_i - v_j - T(p_{ij} - \overline{p}) \leq T\overline{p} - l_{ij}, \quad \overline{p} \in \mathbb{Z}$$

This corresponds to the bounds

$$l_{ji} = T\overline{p} - u_{ij}$$

$$u_{ji} = T\overline{p} - l_{ij}$$

If $0 < u_{ij} \leq T$, we have $p = 1$. If $T < u_{ij} < 2T$, we have $p = 2$. Larger values of $u_{ij}$ do not occur, hence $0 \leq p \leq 2$.

Given a $q_C$ value for each cycle $C \in G$, we need to have

$$\sum_{a\in C^+} x_a - \sum_{a\in C^-} x_a = Tq_C, \quad \forall C \in G$$

where $x_a$ denotes the process time corresponding to that arc. Given the process times, the $q_C$ variables can be calculated as well, since

$$q_C = \sum_{a\in C^+} p_a - \sum_{a\in C^-} p_a$$

Those variables are bounded by $a_C$ and $b_C$ as in Theorem 3.5.

Given the $q$-values for the cycles based on the intervals of the constraints in the cycle, we obtain bounds on the possible $p$-values of the arcs in the cycle. Algorithm 3.4 by Schrijver (2008) can quickly give a timetable if the $p$-values are known. For this algorithm, it limits the amount of possible functions $p : A \to \mathbb{Z}$ for which a function $\pi : N \to \mathbb{Z}/T$ is to be found. If $p = 0$ on the arcs of the spanning tree, the $p$-value for the non tree arc can be calculated, or at least bounds can be given for it.
3.3. PROPERTIES OF THE MODEL

Based on the cycles, PESP can be transformed to the Cycle Periodicity Formulation (CPF) as proposed by Nachtigall (1999):

\[
\begin{align*}
\text{Minimize} & \quad F(x) \\
\text{subject to} & \quad \sum_{a \in C^+} x_a - \sum_{a \in C^-} x_a = Tq_C \quad \text{for all } C \in G \\
& \quad l_a \leq x_a \leq u_a \quad \text{for all } a \in A \\
& \quad a_C \leq q_C \leq b_C \quad \text{for all } C \in G \\
& \quad x_a \in \mathbb{R} \quad \text{for all } a \in A \\
& \quad q_C \in \mathbb{Z} \quad \text{for all } C \in G
\end{align*}
\]  

with some objective function \( F(x) \).

The problem here is that there are too many cycles to check in a practical way, there might be an exponential number of them. This implies an exponential number of constraints and integer \( q \) variables. PESP has only \( |A| \) integer \( p \) variables and \( 2|A| \) constraints. However, there is a way around this. It is sufficient to state the requirement \( a_C \leq q_C \leq b_C \) only for the cycles in a certain cycle basis \( B \) in the cycle space of graph \( G \). This basis has dimension \( c = |A| - |N| + 1 \), much less than the possible exponential number of all cycles in the graph \( G \). Further, it has less integer variables than PESP.

In order to introduce the concept of a cycle basis, some definitions and notations are introduced, they are all taken from Peeters (2003). A cycle basis is defined as

\[
\begin{align*}
\gamma_C &= \begin{cases} 
1 & \text{C contains a in the forward direction} \\
-1 & \text{C contains a in the backward direction} \\
0 & \text{else}
\end{cases} 
\end{align*}
\]  

Now the cycle space of the constraint graph \( G \) is defined as the linear vector space spanned by the vectors \( \gamma_C \) of cycles \( C \in G \). The cycle basis \( B \) is a basis for this cycle space. A basis can be generated as follows:

- Find a spanning tree \( F \)
- Add a non-tree arc \( a \in A \) to \( F \), then a cycle in \( F \) occurs. This cycle is said to be generated by \( a \). Add the cycle vector \( \gamma_C \) to the basis.
- Repeat the procedure for all non-tree arcs, this gives a basis of dimension \( c = |A| - |N| + 1 \).

Now having defined what a cycle basis is and how it can be found, the following definitions are introduced.

**Definition 3.6 (Integral cycle basis).** A cycle basis \( B \) of \( G \) is an integral cycle basis if every non-basic cycle is an integer linear combination of the cycles in \( B \).

Now an important theorem follows that makes the Cycle Periodicity Formulation a lot easier:

**Theorem 3.7.** If the cycle periodicity property

\[
\sum_{a \in C^+} x_a - \sum_{a \in C^-} x_a = Tq_C
\]  

holds for every cycle \( C \) in an integral cycle basis \( B \) of \( G \), then it holds for every cycle in \( G \).

This means that if an integral cycle basis is found, it greatly reduces the number of equations in this reformulation of PESP. Now there are only \( |A| - |N| + 1 \) cycles left to check. The main problem now is in finding a good cycle basis, at least one that it integral.

In order to find a good cycle basis, the definition of a (strictly) fundamental cycle basis is introduced.

**Definition 3.8 (Fundamental Cycle Basis).** A set \( B = \{C_1, \ldots, C_c\} \) of cycles in a graph \( G \) is a fundamental cycle basis if there exists an ordering of the cycles in \( B \) such that \( C_i \setminus (C_{i-1} \cup \ldots \cup C_1) \neq \emptyset \) for \( i = 2, \ldots, c \).

**Definition 3.9 (Strictly Fundamental Cycle Basis).** A set \( B \) of \( c = |A| - |N| + 1 \) cycles in a graph \( G \) is a strictly fundamental cycle basis if there exists a spanning tree \( H \) of \( G \) such that its non-tree arcs generate \( B \).
It is easy to see that each strictly fundamental cycle basis is also fundamental.
Now the following interesting theorem holds:

**Lemma 3.10.** Any non-basic cycle $D$ is a $\{0, \pm 1\}$ linear combination of the cycles in a strictly fundamental cycle basis $B$.

This would mean we need to have a strictly fundamental cycle basis. This can be relaxed a bit, because of the following theorem:

**Lemma 3.11.** Any non-basic cycle $D$ is an integer combination of the cycles in a fundamental cycle basis $B$.

Those two theorems imply the following:

**Lemma 3.12.** Strictly fundamental and fundamental cycle bases are integral.

The above definitions and lemmas show that it is enough to impose the constraint

$$\sum_{a \in C^+} x_a - \sum_{a \in C^-} x_a = Tq_C$$

(3.45)

only for the cycles in the cycle basis that is generated by a spanning tree. Note that the generation of a cycle basis as it was described before does lead to a strictly fundamental cycle basis and therefore this basis is integral. There might be a different cycle basis, since different spanning trees can be found in a graph. Research has been done on this, see for example (Peeters, 2003, Chapter 5.7), Liebchen (2003), Liebchen (2006).

If all cycles in the cycle basis satisfy (3.45), this does not necessarily mean there exists a solution. The problem then is in the combination of multiple cycles.

Consider the graph shown in Figure 3.17. The non-tree arcs are shown dashed. Define the basis cycles as $C_1 = (A, B, C, D, E, F, A)$ and $C_2 = (A, G, F)$. There is also a non-basis cycle, which is $C_3 = C_1 \cup C_2 = (A, B, C, D, E, F, G, A)$. Then

$$a_{c_1} = \left\lceil \frac{12 - 13 + 13 + 13 + 53 + 41}{60} \right\rceil = 2 \quad (3.46)$$

$$b_{c_1} = \left\lfloor \frac{16 - 11 + 14 + 15 + 101 + 43}{60} \right\rfloor = 2 \quad (3.47)$$

and

$$a_{c_2} = \left\lceil \frac{3 - 42 + 41}{60} \right\rceil = 1 \quad (3.48)$$

$$b_{c_2} = \left\lfloor \frac{57 - 40 + 43}{60} \right\rfloor = 1 \quad (3.49)$$
However, we have
\[
\begin{align*}
\alpha_{C_3} &= \left\lceil \frac{12 - 13 + 13 + 13 + 53 + 40 - 57}{60} \right\rceil = 2 \\
\beta_{C_3} &= \left\lfloor \frac{16 - 11 + 14 + 15 + 101 + 42 - 3}{60} \right\rfloor = 2
\end{align*}
\] (3.50) (3.51)

To show how the \( q \)-values of combination of cycles relate together, I state the following theorem:

**Theorem 3.13.** Given a graph \( G = (V,A) \) and two cycles \( C_1, C_2 \in G \) that share at least one arc. Assume that if \( C_1 \) uses the arc in the forward direction, then \( C_2 \) should use it in the backward direction. Then, for the cycle \( C_3 = C_1 \cup C_2 \) we have

\[ q_{C_3} = q_{C_1} + q_{C_2} \]

with

\[ q_C = \frac{1}{T} \left( \sum_{a \in G^+} x_a - \sum_{a \in G^-} x_a \right) \]

where \( x_a \) denotes the process time corresponding to the arc \( a \).

**Proof.** If two cycles in a cycle basis based on a spanning tree share at least one arc, the shared arcs are connected to each other by definition of a spanning tree. Denote the set of shared arcs by \( S \). Merge all the shared arcs into one arc with process time \( x_s \), so

\[ x_s = \sum_{(i,j) \in S^+} x_{ij} - \sum_{(i,j) \in S^-} x_{ij} \]

For both cycles, merge the arcs that are not shared into one arc with process times \( x_1 \) and \( x_2 \), where

\[ x_k = \sum_{(i,j) \in (C_i \setminus S)^+} x_{ij} - \sum_{(i,j) \in (C_i \setminus S)^-} x_{ij}, \quad k = 1, 2 \]

This leads to the situation shown in Figure 3.18.

![Figure 3.18: Simplified cycles](image)

Then \( q_{C_1} = (x_1 + x_s)/T \) and \( q_{C_2} = (-x_2 - x_s)/T \), where the cycles do not use the shared arc in the same direction. Then \( C_3 = C_1 \cup C_2 \) first uses the arc with \( x_1 \). Next, it uses the arc with process time \( x_s \) in both directions. Finally, it uses the arc with \( x_2 \) as process time. Hence the shared arc ‘cancels out’. This means that

\[ q_{C_3} = (x_1 + x_s - x_s - x_2)/T = (x_1 + x_s)/T + (-x_2 - x_s)/T = q_{C_1} + q_{C_2} \]

3.3.6 Total unimodularity

Consider the matrix formulation of PESP. Therefore, define \( l, u \in \mathbb{R}^{|A|} \) as the vectors with the lower and upper bounds respectively. Let \( v \in \mathbb{R}^{|N|} \) be the vector of variables \( v_i \) and \( p \in \mathbb{Z}^{|A|} \) the vector of variables \( p_{ij} \). Define the \( |N| \times |A| \) matrix \( M \) as

\[
M(i,a) = \begin{cases} 
1 & \text{if } a = (i,j) \text{ for some } j \in N \\
-1 & \text{if } a = (j,i) \text{ for some } j \in N \\
0 & \text{otherwise}
\end{cases}
\]
So $M$ is the node-arc incidence matrix. In Cook et al. (1998) an easy to read proof can be found, showing that the node-arc incidence matrix is totally unimodular. Suppose $(v^*, p^*)$ is a solution to PESP, with $p^*$ integer. Since $l$, $u$ and $T$ are integer, the polyhedron
\[ P(p^*) = \{ v \in \mathbb{R}^{|V|} | l - Tp^* \leq M^Tv \leq u - Tp^* \} \]
has integer vertices. Since a solution exists, the solution $v$ is integer. This means it is not necessary to require $v \in \mathbb{N}^{|V|}$ in any solver, those variables can be relaxed to be continuous, in a solution they will be integer, as long as the bounds are integer.

### 3.3.7 Sequencing

Consider two trains sharing the same track. As shown in Section 3.3.5, this resulted in the cycle shown in Figure 3.19.

![Figure 3.19: Two trains share a track](image)

Calculating the lower and upper bounds for the $q_C$ variable in this cycle gives for the lower bound
\[
a_C = \left\lceil \frac{1}{T} \left( \sum_{(i,j) \in C^+} l_{ij} - \sum_{(i,j) \in C^-} u_{ij} \right) \right\rceil = \left\lceil \frac{1}{T} (r^{t_1} + h - r^{t_2} - (T - h)) \right\rceil = \left\lceil -1 + \frac{r^{t_1} + 2h - r^{t_2}}{T} \right\rceil
\]

and for the upper bound
\[
a_C = \left\lfloor \frac{1}{T} \left( \sum_{(i,j) \in C^+} u_{ij} - \sum_{(i,j) \in C^-} l_{ij} \right) \right\rfloor = \left\lfloor \frac{1}{T} (r^{t_1} + T - h - r^{t_2} - h) \right\rfloor = \left\lfloor 1 + \frac{r^{t_1} - 2h - r^{t_2}}{T} \right\rfloor
\]

If $r^{t_1} \leq r^{t_1} - 2h$ for trains $t$ and $t'$, they can pass each other, so this is not allowed. This means that it is required that $r^{t_1} - r^{t_1} < 2h$ to have a safe schedule. So the maximum difference between the possible trip times should not exceed $2h$. It is required to have $a_C \leq q_C \leq b_C$, leading to $0 \leq qC \leq 0$. Therefore, in order to have a safe operation on the tracks, the $q$-value in the cycles that are generated by two trains using the same track should be zero.

### 3.4 The solver CADANS

A description of all the modules of CADANS is given by de Waal (2005). This section is largely based on this work. For the technical reports of the different modules that CADANS consists of, see Odijk, Lentink, Freling, and Steenbeek (2003); Odijk, Lentink, and Steenbeek (2002, 2003a, 2003b, 2003c).
3.4. THE SOLVER CADANS

The solver CADANS has been the most important part of DONS for a long time. In the CADANS program, there are five modules, as shown in Figure 3.1. **Conflex** is the most important of them, since it solves the timetabling problem. If a solution is found, it is ‘optimized’ by the **Optimize** module. To avoid performing a lot of iterations, the **Bepaal Speling** module is used to calculate the flexibility at stations. **Options** is only used when no schedule can be found and to check how this infeasibility can be resolved. **Verify** is used to check feasibility of a found schedule and can also be used to check schedules found by another program.

3.4.1 Conflex

**Conflex** is the module within CADANS that finds a feasible timetable with a time period of $T = 60$. It finds a vector $v : N \rightarrow \mathbb{Z}/T$ on the nodes of the constraint graph. Those nodes are related to each other by constraints that are represented in Conflex as

$$(v_i - v_j) \mod 60 \in S_k \subseteq \{0,1,\ldots,59\}, \quad k = 1,2,\ldots,|A|$$

Conflex basically does three steps: preprocessing, solving and postprocessing. Before describing the solving process, define $M = \{0,\ldots,59\}$ and we call every subset of $M$ a time collection. Let $S \subseteq M$, $R \subseteq M$ and $i \in \mathbb{Z}$. Then we define

$$-S = \{(60 - t) \mod 60 | t \in S\}$$

$$S + i = \{(t + i) \mod 60 | t \in S\}$$

$$S + R = \bigcup_{t \in R} S + t$$

$$S - R = S + (-R)$$

Each constraint $C_k$ belongs to an edge $e_k$ in the constraint graph. Assume that all arcs point from a node with a smaller index to a node with a higher index. Call $C(e_k) \subseteq M$ the set of possible values for that constraint.

**Preprocessing**

In the solving process, first of all the problem is reduced as much as possible, making the solving easier. Some constraints can be implied by others so they can be removed from the graph without loss of generality. It is important to note that if constraints are removed, conflex stores them, so when a minimal conflicting subset of constraints is generated, the original arcs can be shown.

**Elimination of dummy restrictions** Often parallel arcs occur in the graph. Suppose $X$ is a set of parallel arcs and suppose they all point in the same direction, i.e. they have the same head and tail. Arc $e \in X$ is redundant if it is implied by the other arcs in $X$, meaning it imposes no extra restrictions. $C(e)$ denotes the set of possible process times arc $e$ can take, than $e$ is redundant if

$$\bigcup_{a \in X \setminus \{e\}} C(a) \subseteq C(e)$$

If edge $e$ is removed, the other arcs are checked. Conflex removes the arcs in decreasing order of the cardinality of the set of possible process times it can have. This means it starts with $e \in X$ where $|C(e)|$ is the largest.

**Merge and contract** Parallel arcs in the graph are replaced by one single arc that has as set of possible values the intersection of those sets from all the parallel arcs. If this leads to an empty set, the constraints are contradicting. If an arc $e$ results with $|C(e)|$, than this arc can be merged with others. Suppose $e = (a,b)$, it points from node $a$ to node $b$. Than every arc that is connected to node $b$ can be connected to node $a$ instead of node $b$, and the value of the arc can be increased or decreased by $C(e)$, depending on the direction of the arc. If it is in incoming arc, the value is subtracted, else it is added. Doing so, new parallel arcs can arise. Those are merged into one single arc again. An example of merging an arc with $|C(e)|$ is shown in Figure 3.20. The arc between $A$ and $C$ has only one possible value. The result is shown at the right. The arc from $D$ to $C$ is

---

5English: determine margines.
replaced by an arc from $D$ to $A$, with possible values $\{1, 2, 3\} - 5 = \{56, 57, 58\}$. In the resulting graph, parallel arcs arose, so they can be merged into one single arc as well.

![Diagram](image)

**Figure 3.20:** An arc with $l_a = u_a$ can be contracted.

Further, Conflex looks for nodes with degree (sum of incoming and outgoing arcs) of at most two. If a node has degree zero, it can be set to any value, so it can be removed from the graph. If a node has degree one, a value can always be found if the other nodes in the graph are set to some value. It does not give rise to any conflict, so it is removed together with the adjacent edge. In the final phase, it has to be put back again, in order to find the right time in the timetable. Next, nodes with degree two can be removed as well. An example of such a situation is given in Figure 3.21. The two arcs can be merged into a new arc with the sum of the intervals. This means that the possible values for the new arc from $A$ to $C$ is now $\{45, 46\} + \{14, 15\} = \{0, 1, 59\}$ as defined in (3.55). Node $B$ has no influence on other nodes than $A$ and $C$, so given values for those nodes, a value for $B$ can be calculated.

![Diagram](image)

**Figure 3.21:** A node of degree two can be removed.

This procedure is repeated until no parallel arcs are in the graph any more. So the resulting graph is a simple digraph.

**Triangle propagation** Another simplification of the problem is the triangle propagation. This occurs when there is a situation with three nodes that form a clique. An example is shown in Figure 3.22. The arcs $A \rightarrow C$ and $C \rightarrow B$ provide a tighter bound on the arc $A \rightarrow C$ than it currently has. The new interval becomes

$$\{25, 26, 27\} \cap \{15 + 8, 16 + 8, 17 + 8, 15 + 9, 16 + 9, 17 + 9\} = \{25, 26, 27\} \cap \{23, 24, 25, 26\} = \{25, 26\}$$

In this case it does not reduce the number of $p$-values, but there might be situations in which this leads to an arc with a value set of cardinality one, i.e. $l_a = u_a$, leading to a new contraction iteration.

![Diagram](image)

**Figure 3.22:** Triangle propagation.
Mathematically, the following is done. Renumber all the nodes that are present in the graph after the merging and contracting phase, starting with 0. So every node has an index that is smaller than the number of nodes. Further, every arc points from a node with a lower index to a node with a higher index. Define the matrix \(|V| \times |V|\) matrix \(C\) by

\[
C(a, b) = \begin{cases} 
\{0\} & \text{if } a = b \\
C(e) & \text{if there is an edge } e = (a, b) \\
-C(e) & \text{if there is an edge } e = (b, a) \\
\{0, \ldots, 59\} = M & \text{else}
\end{cases}
\] (3.58)

Mark every element \(C(a, b)\) with \(a < b\) where \(C(a, b) \neq M\) as to propagate. Do the following:

1. Choose a marked element \((x, y)\) where \(|C(x, y)|\) is minimal.

2. For all \(z, z \neq x, z \neq y:\)
   - Determine \(C'(x, z) = C(x, y) + C(y, z)\).
   - If not \(C(x, z) \subseteq C'(x, z)\), the other arcs imply a tighter bound for the arc. Then set \(C(x, z) = C'(x, z) \cap C(x, z)\) and \(C(z, x) = -C(x, z)\).
     - If \(x < z\), mark \((x, z)\) or else \((z, x)\) as to propagate.

3. Remove the marker from \((x, y)\).

4. If there are no elements to propagate, stop. Else, repeat the procedure.

For all arcs \(e = (x, y)\), the value for \(C(e)\) is replaced by \(C(x, y)\).

In the example of Figure 3.22, node \(A\) has index 0, \(C\) gets index 1 and \(B\) gets index 2. This leads to the matrix \(C\) as

\[
C = \begin{bmatrix} 
\{0\} & \{8, 9\} & \{25, 26, 27\} \\
\{51, 52\} & \{0\} & \{15, 16, 17\} \\
\{33, 34, 35\} & \{43, 44, 45\} & \{0\}
\end{bmatrix}
\] (3.59)

where the marked elements are shown in blue. Take \((x, y) = (0, 1)\). Then

\(C'(0, 2) = C(0, 1) + C(1, 2) = \{8, 9\} + \{15, 16, 17\} = \{23, 24, 25, 26\}\) (3.60)

Since \(C(0, 2) = \{25, 26, 27\}\), it is not included in \(C'(0, 2)\), so set

\(C(0, 2) = C(0, 2) \cap C'(0, 2) = \{25, 26, 27\} \cap \{23, 24, 25, 26\} = \{25, 26\}\) (3.61)

This leads to

\[
C = \begin{bmatrix} 
\{0\} & \{8, 9\} & \{25, 26\} \\
\{51, 52\} & \{0\} & \{15, 16, 17\} \\
\{34, 35\} & \{43, 44, 45\} & \{0\}
\end{bmatrix}
\] (3.62)

Repeating this leads to no further reductions. The solution is shown in Figure 3.22.

Further, in order to speed up the solving process, a redundant edge between nodes \(a\) and \(b\) is added if:

- There is no arc between \(a\) and \(b\), and
- \(|C(a, b)| < 40\), and
- There is no node \(c\) such that \(C(a, b) = C(a, c) + C(c, b)\).

The triangle propagation is not done for all elements, but only for elements \((x, y)\) where \(|C(x, y)| > g\), where \(g\) is a user defined limit, since this process takes a lot of time.

In Appendix B, the reduction of a small graph is shown in several steps. Some steps are taken bij DONS (removing the arrival nodes), and the other are done in the preprocessing phase of CADANS.
Finally, for each node, the propagation factor is calculated. Define

$$
s_{ci}(x) = \sum_{i=0}^{|V|-1} (60 - |C(x, i)|) \quad \text{for all } x \in V \tag{3.63}
$$

$$
sm_{ax} = \max_{x \in V} s_{ci}(x) \tag{3.64}
$$

Then the propagation factor for a node $x$ is defined by

$$
pf(x) = \frac{100 \cdot s_{ci}(x)}{sm_{ax}} \tag{3.65}
$$

This propagation factor is used in the solving phase, it is used to choose at which node to start the solving.

The merge-contraction phase is repeated once more. After that, it is tried to construct a solution.

**Solving**

In the solving phase, each node is initially given a set of possible values, which is $\{0, 1, \ldots, 59\}$. This is going to be reduced as much as possible until a schedule is found. Sequentially, a node is selected and tried to be fixed to some value. Define for a node $n$ the set $open[n]$ as the set of possible values for that node, and $free(n) = |open[n]|$, so it shows how many elements are possible. A node is chosen that has the highest value for

$$
15 \cdot (60 - free(n)) + pf(n) \text{ if } k(n) \geq 2 \tag{3.66}
$$

plus some user defined value. If the user does not define anything for the importance of some nodes, this extra term is zero. $k(n)$ is the number of active arcs connected to that node. An arc is named inactive if, given the current $open[n]$ sets, it will always be satisfied.

If a node is selected, a certain value in the set of possible values $M$ is chosen, as close as possible to the middle of the interval. Define $x(t) = 1$ if $M = \{0, \ldots, 59\}$. Else, let $x(t) = i \cdot j$, where $i$ is the smallest non-negative number for which $(t - i) \mod 60 \notin M$ and $j$ the smallest non-negative number for which $(t + j) \mod 60 \notin M$. Next, the value $t$ is chosen for which $x(t)$ is maximal. This leads to choosing a value that is as close as possible to the middle of the interval as possible. This value is propagated to other nodes. So assume node $x$ is fixed. Then, for all neighbouring nodes $y$, define $open_{xy} = open[x] + C(x, y)$ and if not $open[y] \subseteq open_{xy}$, set $open[y] = open[y] \cap open_{xy}$. The propagation function either returns that it is possible that this value is propagated (and thus has limited the open[] sets of nodes and has done the propagation), or returns a node $n$ where $free(n) = 0$ because of the propagation and undoes the propagation. If the propagation is possible, add the node with the set value to the set of nodes for which a value is found. If for no value it is possible to propagate it, we have a conflict, given the set of fixed nodes. The set of fixed nodes and the set of nodes where the propagation got stuck is send to a function that tries to repair it by changing the times for the events in the set of fixed nodes.

If the solution has to be repaired locally since propagation got stuck, define $X$ as the set of nodes where the propagation got stuck and add the node from which the propagation was done to it. Try to solve this set. If this is possible, add some nodes from the environment of $X$ that are contained in $S$ to it, where the environment means that the nodes are close to $X$ in the constraint graph and try to solve this larger collection. If this is possible, the node from which the propagation was done can be added to the fixed nodes since it is possible to find a value for it. If it is not possible, add some more nodes from the fixed nodes to the environment and try to solve this larger collection. This means some more nodes that were fixed are revisited to change the value. Solving the separate parts means that the solving every time restarts, in order to find a solution. If no conflict and no solution is found in the time that was allowed for the solver, the node from which the propagation was done is added to a set of ‘hard nodes’. Then the set of fixed nodes might be changed again.

Solving subsets of the graph as mentioned in the description above is done in an iterative way. Set the dead end counter for each node to zero. Pick a certain node from the graph, give it a value, and propagate this value. Choosing the node is done by calculating

$$
(60 - free(n)) + \frac{pf(n)}{20} + 100 \cdot \frac{dead\_end\_counter[n]}{15 + dead\_end\_counter[n]} \text{ if } k(n) \geq 2 \tag{3.67}
$$
and choosing the node $n$ with the highest value for this. If no conflict is found for this value in the set time, this node is no dead end and we solve the remaining nodes of the graph given this set value and repeat this procedure. If a conflict is found given this set value, change the value if possible and redo the solving. If no value leads to a solution, increase $dead\_end\_counter$ for this node. The node is a dead end, meaning that the value of other nodes have to be changed, this is the backtracking part. In the end, we will find a solution or a conflict which is send to the post processing phase.

**Postprocessing**

In case a solution is found, all nodes are fixed. The other nodes that were removed in the preprocessing phase are given a time set of $\{0, 1, \ldots, 59\}$. Then they are fixed using the adjacent arcs. This can not cause any conflicts. A final schedule is returned to the user (in fact it is returned via the optimize module).

In the solving part, the solver either found a solution, a conflict, or a time limit was exceeded. In case of the conflict, is could not solve a set of nodes $K$, so the problem is in this set. To compute a minimal set of constraints, the algorithm tries to remove a node $n$ from $K$. If this still gives a conflict, node $n$ was not needed for the conflict, so we set $K = K \setminus \{n\}$, remove the adjacent arcs and repeat the procedure. This leads to a node-minimal set. Next, we do the same for the constraints. First, we build the graph that contains the nodes of the conflict and all arcs between those nodes that were in the original graph. Since the set must be minimal, removing one arc will solve the problem. Hence, we remove an arc and see if we still have a conflict. If not, this arc has to be in the minimal conflict set. Else, try another arc until the minimal set is found.

### 3.4.2 Optimize, Bepaal speling, Options and Verify

The remaining modules do only manipulate the output of conflex or are used to support it in finding a schedule. They are not used to generate a timetable themselves.

**Optimize**

Conflex does only look for a feasible schedule. Once a feasible schedule is found, it stops. This schedule is sent to the Optimize-module, to do some optimization. The only thing this module does is moving the found event times away from undesirable positions. Consider a line driving with a frequency of two. Then those lines preferably travel in a 30/30 pattern. The constraint imposing this, where a bandwidth of 2 is used, is

$$28 \leq t_2 - t_1 + 60p_1 \leq 32$$

Optimize tries to move the time difference between $t_1$ and $t_2$ as close to 30 as possible, to have the best pattern. This is done for all the constraints, by penalizing undesirable positions and minimizing a weighted sum of this. For this procedure CPLEX is used.

It is important to note that optimize does not generate a timetable from scratch, but uses the output from the conflex-module, with fixed $p$-values.

‘Bepaal speling’

Determining a timetable is a tough job. Given a timetable, this does not necessarily have to be feasible for the stations, since they are considered to be black boxes with unlimited capacity. Whether it fits can be checked by the station module. However, most of the times this will not fit. In order not having to perform a lot of iterations before a feasible schedule is found, the bepaal speling module is used. This calculates the magnitude of the margins of the departure times at the stations. This can be done in two ways. First of all, it can be calculated how much a certain point in time can be moved. The other option is to calculate how much a point of time can be moved when other times are allowed to move as well, as long as the structure of the timetable is preserved. The user can decide on this. The last option has more flexibility in it, but is costs more calculation time.
Options

When no feasible schedule is found, conflex returns a minimal set of conflicting constraints. Here the options module can check how the infeasibility can be resolved for this conflict. It calculates for each constraint in the conflict and for each bound of a constraint how much it has to be changed in order to get feasibility within the set of conflicting constraints. When an interval is widened to \([0, 1, \ldots, T-1]\), all values are possible, and because of the fact that the conflicting set is minimal, the conflict is removed. Note that this module only takes into account the constraints that are within a conflict. It might be possible to change a bound in order to resolve a conflict, without actually solving it since another constraint might just be parallel and preventing the existence of a feasible solution.

In the version of DONS that is used for this research, this module is not found, so only a minimal conflicting subset of constraint is returned. It was possible to call the options-module separately to check how the constraint bounds had to be relaxed.

Verify

This module is only used to check a derived timetable for feasibility. It is especially used when a timetable is generated by another program than conflex or optimize and this timetable is to be checked for feasibility.
Chapter 4

Solution

One of the shortcomings that CADANS has, is that it does not come up with a timetable when conflicts are found. Next to that, it is hard to find out why the conflict is there and how to solve it. In this chapter, an approach is described that solves this problem. It finds a minimal relaxation or change of the constraints, according to some objective function, that solves the infeasibility that is found by CADANS.

4.1 Introduction

As was described in chapter 3, a PESP constraint is a constraint of the following type:

\[ l_{ij} \leq \pi_j - \pi_i + T_{p_{ij}} \leq u_{ij} \iff \pi_j - \pi_i \in [l_{ij}, u_{ij}] \text{.} \]  

(4.1)

This means that the difference between two event times should be within a certain (periodic) interval.

In the case that CADANS can not find a timetable but returns a conflict, it consists of a set of such constraints. The bounds for those constraints are put too tight for a solution to exist. The size of this set is minimised in such a way that skipping one of the constraints would solve the conflict. Hence, to find a timetable one might delete a constraint in the conflict and check whether there are other conflicts. This procedure can be repeated until no conflicts are arising anymore. Afterwards, planners can check why the constraint had to be skipped or what other method is possible to solve the problem.

In the approach of deleting constraints, a choice has to be made which constraint to skip. All constraints in the model are required with a certain reason. Some of them will be very important and cannot be left out, like safety constraints. Others are requirements that the management has put to the model, like providing good connections between trains or serving lines in a nice pattern, like having 30 minutes between trains. Those constraints could be skipped without influencing the model much, at least not the safety aspect. The choice on the constraint that is skipped greatly influences the running time of the suggested approach. A constraint might be deleted that solves more conflicts in one go. Therefore, it is impossible to state how many constraints have to be skipped in total in a timetabling problem. On the one hand, it is not known how many conflicts there are. On the other hand, the way a conflict is solved influences the other conflicts.

As mentioned already, skipping a constraint might cause an undesired situation. If a trip time constraint is deleted, it means that a certain line is cut into pieces. In this case it is better to delay a train in stead of cutting the line into two pieces. If a safety constraint is skipped, everything might fit on the tracks, but safety is not guaranteed. However, in this case it can be checked why safety constraints had to be violated and how other things like triptimes or frequencies can be changed such that safety is guaranteed again.

Although this procedure might go in a satisfactory direction, it might be better to relax or change constraints instead of deleting them. Constraints should be relaxed in such a way that the conflict is solved and the constraints are relaxed as little as possible. Often, only a slight adaptation of the constraints is sufficient to solve the conflict, like changing a bound by only one minute. This leads to solving a minimisation problem to find the smallest adaptation to solve a conflict, but it leads to acceptable solutions in the end.
CHAPTER 4. SOLUTION

This method is the core of the solution approach that is followed in this thesis. If CADANS finds a conflict in the constraints, the constraints are adapted in such a way that the conflict is solved. Choosing the constraints to adapt is very important. A choice can be made that solves the conflict, but might lead to other problems. It would be better to find such a solution that is not only good to solve this conflict, but also leads to no new conflicts. This means as much information from the ‘neighbourhood’ of the constraint graph should be taken into account that provides relevant information to the problem. One should notice that taking into account too much information might lead to long computation times. The method extra information can be taken into account can be done in several ways, as will be shown later in this chapter.

To solve a conflict, a Mixed Integer Problem is solved. Slack variables are added to the lower and upper bound of the constraints that can be relaxed or changed and the weighted sum of those slack variables is minimised. Constraints that can be relaxed are for example trip time constraints or connection constraints. Constraints that can not be relaxed are safety constraints. However, if trip times are changed, often the safety constraints have to change as well, since they depend on the trip times, as is shown in the previous chapter.

4.2 Relaxation of constraints

A conflict consists of a set of nodes (events) and arcs between those nodes (constraints) that cannot be satisfied at the same time. In order to derive a mathematical model to solve such a conflict, define the following sets:

\[ N_C = \{ \text{nodes involved in conflict} \} \subseteq N \]  \hspace{1cm} (4.2a)
\[ A_C = \{ \text{arcs involved in conflict} \} \subseteq A \]  \hspace{1cm} (4.2b)

This leads to a sub graph \( G_C = (N_C, A_C) \) for which it is known that no solution exists, i.e., there is no feasible periodic potential \( \pi : N_C \rightarrow \mathbb{Z}/T \) such that it satisfies all the constraints. Here a periodic potential is defined as follows:

**Definition 4.1 (Periodic potential, Peeters (2003)).** A function \( \pi : N \rightarrow \mathbb{R} \) is a periodic potential with period \( T \) if \( 0 \leq v_i \leq T - 1 \) for all \( i \in N \).

In order to make a distinction between the arcs that can be changed and those that cannot be changed, define the following:

\[ A^f = \{ \text{flexible constraints} \} \]  \hspace{1cm} (4.3a)
\[ A^* = \{ \text{fixed constraints} \} = A \setminus A^f. \]  \hspace{1cm} (4.3b)

So \( A^f \) contains all the arcs from the conflict that can be relaxed. Constraints for which the bounds can be relaxed are trip time, dwell time, connection, frequency or absolute constraints. Safety constraints can not be relaxed and hence they will be contained in \( A^* \). However, note that they might change because of changes in the constraints in \( A^f \).

As mentioned in the previous chapter, there are several types of constraints. They are identified with one or more letters. Let \( T \) be the set of identifiers for the constraints. Then we have

\[ T = \{ D, S, F, A, ABS, FB, UI, IU, II, UU, H \} \]  \hspace{1cm} (4.4)

The meaning of those letters for a constraint are as follows:

- \( D \) passing constraint (doorkomst)
- \( S \) stop constraint
- \( F \) frequency between trains
- \( A \) connection (aansluiting)
- \( ABS \) fixed (absolute) time
- \( FB \) frontal collision (frontale botsing)
- \( AR \) overtaking (achterop rijden)
- \( UI \) out-in relation
- \( IU \) in-out relation
- \( II \) in-in relation
- \( UU \) out-out relation
- \( H \) hinder

(4.5)
4.2. RELAXATION OF CONSTRAINTS

Now $A_{\tau}, \tau \in T$, means that it contains all the constraints of type $\tau$. Note that

$$A_D \cup A_S \cup A_F \cup A_A \cup A_{ABS} \subseteq A^f \tag{4.6}$$

$$A_{AR} \cup A_{FB} \cup A_{FI} \cup A_{IU} \cup A_{II} \cup A_{UU} \cup A_H \subseteq A^* \tag{4.7}$$

This clearly shows that most types of constraints are on safety. The constraints concerning passings and stops define the train lines. They are separated in time by safety constraints. In general, a lot of safety constraints come into play when a single track is considered, but also for other parts a lot of safety constraints are required.

4.2.1 Relaxation

Let $h, t : A \rightarrow N$ be the ‘head’ and the ‘tail’ of a constraint respectively, as it can be seen in the constraint graph, so $h(a) = v_j$ and $t(a) = v_i$. Now (4.1) can be written as

$$l_a \leq \pi h_a - \pi t_a + T p_a \leq u_a \tag{4.8}$$

Note that this constraint is composed of the following two constraints:

$$l_a \leq \pi h_a - \pi t_a + T p_a \tag{4.9a}$$

$$\pi h_a - \pi t_a + T p_a \leq u_a \tag{4.9b}$$

Relaxing the constraints means providing more flexibility for the difference in time between two events. Hence, the lower bound should be decreased and the upper bound increased. It is not desirable to relax all constraints, but to find the smallest change that is possible in order to obtain a feasible solution. For each arc $a \in A^f_C$, define slack variables $s^l_a, s^u_a \geq 0$ and rewrite the constraints to

$$l_a \leq \pi h_a - \pi t_a + T p_a + s^l_a \tag{4.10a}$$

$$\pi h_a - \pi t_a + T p_a - s^u_a \leq u_a \tag{4.10b}$$

Those slack variables account for the change in the bounds of the constraint. Having a feasible solution in the conflict means the solution satisfies all the (relaxed) constraints. Now minimizing a weighted sum of the slack variables means finding a feasible solution that satisfies constraints that deviate as little as possible from the original constraints.

Minimizing the (weighted) sum of the slack variables means finding the smallest adaptation of the conflicting constraints that would solve the conflict. Define the set of slack variables as

$$S = \bigcup_{a \in A^f} \{s^l_a, s^u_a\} \tag{4.11}$$

Then the objective can be defined as

$$\text{Minimize } \sum_{s \in S} w_s \cdot s \tag{4.12}$$

4.2.2 A further look into the flexible constraints

When a constraint is relaxed and an optimal solution is found to the minimisation problem, either the upper slack variable or the lower slack variable will be non-zero, never both of them. Suppose they would both be non-zero. The difference in time between the events $i$ and $j$ is $\pi_j - \pi_i$. Then, by definition of minimality of the solution that is found, this difference should be equal to $l_{ij} - s^l_{ij}$ and $u_{ij} + s^u_{ij}$. However, since $l_{ij} \leq u_{ij}$, and hence $l_{ij} - s^l_{ij} < u_{ij} + s^u_{ij}$, this can not be true. So at most one of the slack variables can be non-zero.

Consider trip time constraints, those are the constraints of type $D$ and $S$. DONS assumes a fixed trip time so the constraint corresponding to that has the same lower as upper bound. If one of those bounds is changed, in fact a flexible trip time is introduced. However, this introduces some difficulty with the safety constraints that depend on the trip time, as is described in the next subsection. It is possible not to relax the trip time constraint, but to change it, i.e. change the lower and upper bound simultaneously. Suppose the trip time constraints are relaxed, many safety constraints have to be changed as well. It is possible to consider the worst case scenario for the
safety constraints, i.e., consider the situation that gives the tightest bounds. However, in doing so it can lead to the situation in which certain tracks are ‘too safe’, trains are separated from each other by more minutes than is necessary. This might lead to the problem becoming infeasible, especially when single track is considered.

This problem can be overcome by changing the current PESP model to a model that includes arrival events as well. Here it is often possible to allow for some flexibility in the trip times, as is shown in Kroon and Peeters (2003). However, since this implies quite a deviation from the model that is used by DONS and CADANS and it means basically doubling the number of events in the network (and hence considerably more constraints), this is not done here. It would mean that CADANS and DONS have to be changed, which is out of scope for this thesis. Therefore, trip times are assumed to be fixed. If they are changed, the lower and upper bound for the constraint are changed simultaneously. In this way, the safety constraints can be changed in a clear way as well. By including more information about the problem than only the conflicting constraints, i.e., add more constraints to the conflict, it is possible to obtain satisfactory solutions, so this does not impose any restrictions. So in this thesis, the assumption of fixed trip times still holds. Trip time constraints are changed instead of relaxed, the lower and upper bound are changed simultaneously.

4.2.3 Dependent constraints

As is shown in Section 3.2.5, safety constraints might depend on the trip time associated with the trains that are involved in this constraint. Also, connection constraints might depend on the trip time of some train on a certain track.

As stated above, trip times are allowed to be changed, and hence the bounds for the constraints are not relaxed but changed simultaneously. However, if adaptations to the constraints are to be found, one needs to have the right safety constraints. Therefore, if those constraints depend on the trip time, and the trip times get some slack variables to allow for changes, those slack variables need to be incorporated in the bounds of the dependent safety or connection constraints as well.

For a trip time constraint, one has that 
\[ l_a + s_{a_2} + r_a = u_a - s_{a_1} \]
In the following it is described how the constraints that depend on the trip time are changed.

Frontal collision

If the trip time constraint on single track is adapted, this changes the constraint preventing frontal collisions. The trip time for both an incoming and an outgoing train can be changed. Consider a frontal collision constraint \( a' \in A_{FB} \). A sketch of a time space diagram is shown in Figure 4.1 where the arc \( a' \) preventing the frontal collision is shown dashed. As mentioned in section 3.2.5 this arc contains two components, depending on the order of the trains. Therefore, two arcs are shown, the left taking care of the lower bound, the right taking care of the upper bound.

![Figure 4.1: Overview FB-situation](image)

If \( t_a = t_{a'} \) for \( a \in A_D \cup A_S \) (left arc \( a' \)), an outgoing train from station \( s \) is considered. In that case, the lower bound for \( a' \) has to be changed to \( l_{a'} + s_{a_1} = l_{a'} + s_{u_a} - s_{l_a} \).

If \( t_a = h_{a'} \) (right arc \( a' \)), an incoming train at \( s \) is considered. In that case, the upper bound has to be decreased to \( u_{a'} - s_{a_2} = u_{a'} - s_{u_a} + s_{l_a} \).
4.2. RELAXATION OF CONSTRAINTS

Connection constraints and in-out relations

In the case that a trip time of a train is changed which serves as a feeder for an other train, the connection constraint has to be changed as well. This situation is sketched in Figure 4.2. The trip and dwell arcs for train $t_1$ are shown in blue, for train $t_2$ in red. There is a connection from $t_1$ to $t_2$ which is shown dashed.

This means that if for some $a' \in A_A$ one has that $t_a = t_{a'}$ for $a \in A_D \cup A_S$, than the lower and upper bound for $a'$ both increase by the increase in trip time, so both are increased by $s_{a'} - s_a$.

The same holds for in-out relations as can be seen in Figure 4.1. It creates the same situation as a connection between train. The bounds for this constraint depend on the trip time of the incoming train. So all arcs in $A_{IU}$ are changed in the same way.

Out-in relations

For out-in relations, the constraints depend on the trip time of the incoming train as well, but now in another way. If the trip time of the incoming train is increased, the bounds for the out-in constraints have to be decreased, since both the lower and the upper bound have the term $-r_2$ in them, as seen in (3.17). So, if $t_a = h_{a'}$ for $a' \in A_{UI}$ and $a \in A_D \cup A_S$, then both the lower and the upper bound for $a'$ are decreased by $s_{a'} - s_a$.

In-in relations

In-in relations depend on the trip time differences between two trains, see (3.15). So if one or both of those trip times are changed, than the in-in constraint is changed as well. If $t_a = t_{a'}$ for $a' \in A_{II}$ and $a \in A_D \cup A_S$, then both the lower bound and upper bound are increased by $s_{a'} - s_a$. If $t_a = h_{a'}$, then both the lower bound and upper bound are decreased by $s_{a'} - s_a$. So both slack variables for the trip times (if they are involved in the conflict) are to be added in order to guarantee the right safety constraints.

Hindering constraints

As is described in the previous chapter, conflicts can occur within train path points. For small points, the passing time through such a train-path points is zero. Therefore, this constraint can be stated in departure events from this points, so no trip times are included. Hence, this constraint is fixed.

However, for larger train-path points, like stations, those constraints were similar to the in-out constraints. Therefore, trip times have to be taken into account in the same way as for those type of constraints.

In order to determine the type of constraint that is stated by DONS, it is necessary to check the nodes/events that correspond to the constraint. Each event corresponds to a trip of a train, from one place to the other. The following situations are possible:

- If for both events the departure station is the same, it means that two trains leave the same point. Hence, no trip times are involved and the constraint has to be stated as it is.
- If for both events, the arrival station is the same, it means that two trains enter the same train-path point, it is an in-in relation. Here, trip times of both the trains are included and the same slack variables as for the in-in relation are to be included.
- If one train enters and the other leaves, the arrival station of one event is the departure station for the other event. Now only the slack variables of the entering train are to be included.
Overtaking constraints

For larger trip time differences, the constraint that prevents overtaking is added. This depends on the trip time differences as well. The constraint is stated as

\[ d^t_n - d^t_n \in [r_t - r_{t'} + h, T - h] \quad (4.13) \]

In the DONS system, this constraint is put differently, namely

\[ d^t_n - d^t_n \in [h, T - r_t + r_{t'} - h] \quad (4.14) \]

This means that for each \( a' \in A_{ar} \), slack variables coming from \( a \in A_D \cup A_S \) are added if \( t_a = h_{a'} \) or \( t_a = t_{a'} \). If \( t_a = t_{a'} \), the upper bound is decreased by \( s^u_n - s^l_n \). If \( t_a = h_{a'} \), the upper bound is increased by \( s^u_n - s^l_n \).

### 4.3 A mathematical model

Each node in the PESP constraint graph denotes a departure of a train from some train path point. Most of them has a trip time arc attached to it, pointing to the departure of the train on the following station. Only the last departure of the train has no next departure and hence has no trip time arc corresponding to it. Let \( d : N \rightarrow A_D \cup A_S \) be a function from a node to the corresponding trip time arc.

In order to distinguish the different types of constraints corresponding to hinderings, the following subgroups are introduced:

- \( H_{II} \), for hindrance between two entering trains.
- \( H_{IU} \), for hindrance if the first train is entering and the second is leaving.
- \( H_{UI} \), for hindrance if the second train is entering and the first is leaving.
- \( H_{UU} \), for hindrance between two outgoing trains.

In the following, the subscript \( C \) to make clear the the arcs are involved in the conflict is omitted for readability. The mathematical model to find the minimum adaptations for a feasible solution to exist for any PESP constraint graph \( G = (N, A) \) can now be written as follows:

Minimise \( \sum_{s \in S} w_s \cdot s \)

Such that

\[
\begin{align*}
\pi_{ha} - \pi_{ta} + Tp_a + s^l_n \geq l_a & \quad (\forall a \in A^f \setminus A_A) \\
\pi_{ha} - \pi_{ta} + Tp_a - s^u_n \leq u_a & \quad (\forall a \in A^f \setminus A_A) \\
\pi_{ha} - \pi_{ta} + Tp_a + s^u_n \geq l_a + s^l_{d(t_a)} - s^l_{d(t_a)} & \quad (\forall a \in A_A) \\
\pi_{ha} - \pi_{ta} + Tp_a - s^u_n \leq u_a + s^l_{d(t_a)} - s^l_{d(t_a)} & \quad (\forall a \in A_A) \\
\pi_{ha} - \pi_{ta} + Tp_a \geq l_a + s^l_{d(t_a)} - s^l_{d(t_a)} & \quad (\forall a \in A_{FB} \cup A_{UI} \cup A_{HI}) \\
\pi_{ha} - \pi_{ta} + Tp_a \leq u_a - s^u_{d(h_a)} + s^l_{d(h_a)} & \quad (\forall a \in A_{FB} \cup A_{UI} \cup A_{HI}) \\
\pi_{ha} - \pi_{ta} + Tp_a \geq l_a - s^l_{d(h_a)} + s^l_{d(h_a)} & \quad (\forall a \in A_{IU} \cup A_{H_U}) \\
\pi_{ha} - \pi_{ta} + Tp_a \leq u_a + s^l_{d(t_a)} - s^l_{d(t_a)} & \quad (\forall a \in A_{IU} \cup A_{H_U}) \\
\pi_{ha} - \pi_{ta} + Tp_a \geq l_a + s^l_{d(t_a)} - s^l_{d(t_a)} - s^u_{d(h_a)} + s^l_{d(h_a)} & \quad (\forall a \in A_{II} \cup A_{HI}) \\
\pi_{ha} - \pi_{ta} + Tp_a \leq u_a + s^l_{d(t_a)} - s^l_{d(t_a)} - s^u_{d(h_a)} + s^l_{d(h_a)} & \quad (\forall a \in A_{II} \cup A_{HI}) \\
\pi_{ha} - \pi_{ta} + Tp_a \geq l_a & \quad (\forall a \in A_A) \\
\pi_{ha} - \pi_{ta} + Tp_a \leq u_a - s^u_{d(t_a)} + s^l_{d(h_a)} & \quad (\forall a \in A_{AR} \cup A_{UU} \cup A_{HU}) \\
\pi_{ha} - \pi_{ta} + Tp_a \leq u_a & \quad (\forall a \in A) \\
p_a \in \mathbb{Z} & \quad (\forall a \in S) \\
l_a, u_a \in \mathbb{N}_{\geq 0} & \quad (\forall a \in S) \\
s \geq 0 & \quad (\forall s \in S) \\
\pi : N \rightarrow \mathbb{Z}/T & \quad (4.15)
\end{align*}
\]
4.4 Solving the model

The model just given is a model, where all variables are required to be integer. Hence, this is an Integer Program, which can be solved by a branch and bound algorithm.

The constraints in this model are ‘big $M$’ constraints because of the term $T_p$ involved. This makes optimisation very hard. In fact this model is an extension of PESP (setting all slack variables to zero will return the original PESP) and therefore it is NP-complete as well. This means that the problem instances that are solved by this model should not be ‘too large’, since the solving time might grow exponentially in the size of the instance that is solved. However, there are certain methods that might speed up the process a bit.

First of all, some notes about the conflict graph will be mentioned. Afterward, some tricks will be given that might speed up the process of solving.

4.4.1 On the conflict graph

The conflict that CADANS might return, is a subgraph $G_C = (N_C, A_C)$ for which no solution exists and in which the number of arcs is minimized, such that skipping one of them solves the conflict. Note that it is possible that the same conflict can be given, while using less constraints.

To demonstrate this, consider the situation shown in Figure 4.3.

Figure 4.3: Example of a situation showing that a conflict might contain less arcs

The graph in this figure has no solution. When the size of this conflict is minimised by CADANS, the algorithm tests whether there is still a conflict if some arc is removed. If first the arc between nodes 1 and 2 is removed, there is still a conflict, containing four arcs in total. However, if first the arc between nodes 1 and 4 is removed, the arc between nodes 4 and 2 can be removed as well, while still having the conflict. However, this conflict involves only three arcs. Hence, the conflict that is returned does not necessarily contain the least number of arcs possible, but it is minimal in the sense that if one of the remaining arcs is removed, the conflict is removed.

Solving the model as shown in (4.15) will solve this conflict, but it might make a choice that is not optimal, knowing more about the other constraints in the graph around the conflict. It might easily lead to another conflict, thus postponing the problem. Further, it is possible that there are more similar arcs preventing the existence of a feasible solution, while only one of them is shown in the conflict, since the conflict size is minimized. This is the situation as is shown in Figure 4.3. If arc $(1, 2)$ is relaxed such that the bounds become $[5, 10]$ and arc $(3, 2)$ is relaxed to $[5, 10]$, there is still a conflict since the arcs $(1, 4)$ and $(4, 2)$ prevent feasibility.

Therefore, it might be better to add some more arcs to the conflict graph $G_C$ than just the arcs in the conflict. This increases the size of $G_C$ and hence the size of the model to solve grows, but it might provide more information to make a better choice in the solving of the conflict. This has to be done in a clever way, such that the ‘right’ arcs are added that contribute to the problem in a significant way. This can be done in several ways, as will be described here.

- One thing that can be done is to add all arcs for which both head and tail are involved in the conflict, i.e. all $a \in A$ for which $h_a, t_a \in N_C$. This is what is referred to as ‘adding the intermediate arcs’. Those arcs provide more relations within the nodes of the conflict, making it probably lead to better solution. Depending on the structure and the size of the (conflict) graph, this might lead to a large amount of extra arcs. Sometimes it might add over 2000 extra constraints.

- Another option, which adds some more arcs than the previous option, is to add all incoming and outgoing arcs. At least all arcs of the previous option are included. Further, it might
happen that some arc is put between \( v_1 \in N_C \) and \( u \notin N_C \), while another constraint is put between \( v_2 \in N_C \) and \( u \). This provides another relation, via \( u \), between \( v_1 \) and \( v_2 \). For the other arcs, they might connect to some node that gets a total degree of one in the final conflict graph \( G_C \), thus making it easy to deal with in the MIP-solving process. Furthermore, it is also possible to remove those arcs, since they do not provide any new information.

- The conflict involves certain train series with a certain direction. Knowing this, some extra trip around the point where the conflict occurs can be added, e.g. three trips before and after the trips that are involved in the conflict. This gives more information on the geographical neighbourhood of the conflict. For this, it is necessary to add also arcs that are between the newly added nodes, like all intermediate arcs, otherwise it will serve as a ‘dead end’ and it gives no extra relevant information.

If in this thesis is spoken about adding train series arcs, it means that the trip time constraints are added that correspond to the three trips the train makes before and after the trips in the conflict.

- Single track situations easily give rise to conflicts. Not all train series using this single track have to be involved in the conflict. However, in order to make an effective solution, it is better to include all trains that use this infrastructure. It can easily be checked if single track is involved, since that would mean having a constraint of type \( FB \) attached to at least one of the nodes in the graph.

Note that any combination of the points mentioned above is possible. Once having added all incoming and outgoing arcs, this procedure can be repeated. Further, it is possible to add more trips of a train involved in the conflict. The number of trips may vary, depending on what is best. Also all intermediate arcs can than be added.

There is one important observation to make. Many constraints, especially safety constraints, depend on the trip time of a train on a part of the network. This means that if some trip time arc is changed, all those constraint have to be added. It would be possible to solve the MIP-model and afterwards change those arcs. Another way to deal with this is to add the dependent arcs to the model. If they form ‘dead ends’, giving no extra relations, they will not really influence the calculations. If they connect to other nodes in the graph, they might give more information. In this way, it is made sure that the dependent constraints will be processed correctly and to avoid getting other conflicts if they are not taken into account.

### 4.4.2 Use total unimodularity

As mentioned before, the model in (4.15) is an IP-model, all variables are required to be integer. This might lead to a lot more branches in the search tree of the branch and bound algorithm. If some variables can be relaxed to be continuous, the algorithm will not branch on those variables and hence this might speed up the whole process. Note that in Section 3.3.6 it was mentioned that in the matrix formulation, the coefficient matrix was totally unimodular. This meant that is was not necessary to require that the event times are integer, i.e. \( \pi : N \rightarrow \mathbb{Z}/T \), but we can now say that they can just be real numbers, i.e. \( \pi : N \rightarrow \mathbb{R}/T \). Note that there are slack variables added in this model, but as long as they are required to be integer, the relaxation will result in integer event times.

### 4.4.3 Adding cuts

When CPLEX (or other MIP solvers) starts its solving procedure, it first of all tries to add extra cuts to the model in order to speed up the whole process. Providing some more knowledge about the structure of the constraint graph might be beneficial. This might cut away certain solution, at least in the LP relaxation, hence limiting the search tree and speeding up the procedure. This can be done in the case of this model as well. It is possible to add some cuts as that limiting the possible \( p \)-values. Note that it is not known whether this will give an improvement. For some cases, it works, for other it does not help anything. If a feasible solution is found, it has to satisfy the extra cuts, so a MIP solver determines those relations itself as well.

As mentioned in Theorem 3.5 we have for each cycle \( C \in G \) that

\[
\sum_{(i,j) \in E^+} p_{ij} - \sum_{(i,j) \in E^-} p_{ij} \leq b_C
\]  

(4.16)
where

\[
\begin{align*}
 a_C &= \frac{1}{T} \left( \sum_{(i,j) \in C^+} l_{ij} - \sum_{(i,j) \in C^-} u_{ij} \right) \\
 b_C &= \frac{1}{T} \left( \sum_{(i,j) \in C^+} u_{ij} - \sum_{(i,j) \in C^-} l_{ij} \right)
\end{align*}
\]

There might be an exponential number of cycles in the constraint graph. However, as is described in Section 3.3.5, it is enough to state the cycle periodicity constraint for the cycles in a cycle basis. Such a basis can be found using a spanning tree in the graph. Finding a spanning tree in the graph can be done in polynomial time by Prim’s algorithm for example. Based on this tree, a cycle basis can be found. This is done by adding a non-tree arc to the tree and finding a path in the tree between the nodes between which the extra arc is stated. Now some cuts can be added to the MIP by stating (4.16) for each cycle in this basis. Note that it is important to take into account the allowed slack by calculating the bounds in (4.17), so one should take the largest possible bounds, in order not to cut off parts of the solution space that might contain the best solutions. It is also possible to find more cycles and hence adding more cuts to the MIP-problem, using more cycles than those in the cycle basis.

The spanning tree can be chosen in several ways. When the weight of an arc is defined as the difference between its maximum possible value for the upper bound and the smallest possible value for the lower bound, one can find a minimum weight spanning tree. This contains all arcs which allow only for limited flexibility.

Further, it is possible to limit the \( p \)-variables to \( p \in \{0, 1, 2\} \). Other situations will never occur since for all \( a \in A \) it is known that \( l_a \in [0, T) \) and \( u_a - l_a < T - 2 \). Therefore, more than two cycle shifts will never occur. For each arc, the possible \( p \) value can be determined. Often one has \( p \in \{0, 1\} \), limiting the possibilities even further.

4.4.4 Branching strategy

In a branch and bound algorithm, the branching strategy is very important to find good solutions in short time. Branching on the ‘wrong’ variables consumes a lot of time and one hardly gains anything by doing so. Since the \( p \)-variables determine the structure of the graph, it might be beneficial to branch first on those variables.

All \( p \)-variables might be equal, but some might be more important than others to branch on first. Note that the constraint graph is often very dense, the number of constraints exceed the number of nodes by far. Therefore, it might be beneficial to fix one of the nodes to zero, thus cutting a way a lot of symmetric solutions. Note that shifting each event by one minute gives a feasible schedule again. Therefore, fixing the nodes with the highest total degree (number of incoming and outgoing arcs) to zero, sets the basis for the structure of the solution. In order to give appropriate branching priorities for the other arcs, initialise \( F = \emptyset \). Next, add the fixed node to \( F \). Set a counter to a high value, for example \( c = 1000 \). Do the following:

- \( A = \{ \text{all incoming and outgoing arcs of } F \} \).
- For all arcs in \( A \), if head and tail are both in \( F \), assign its \( p \)-variable priority \( c - 1 \), else \( c - 2 \). Add both of its nodes to \( F \).
- Set \( c := c - 2 \).
- Repeat this procedure until all arcs are processed.

This gives a kind of circular branching priority, starting from the fixed node and going away in it in a circular manner. For the arcs around the fixed node, mostly only one possibility for \( p \) is left. During development of this method, this setting appeared to be useful for large MIP’s. For smaller problems, it might not be beneficial, that will be tested.
4.4.5 Using special order sets

In (4.15), slack variables $s_a \in \mathbb{N}$ were added to take care of the relaxations of the bounds for the constraints. Those are required to be integer. Suppose that $0 \leq s_a \leq k$, then it can also be written as

$$s_a = \sum_{i=0}^{k} i \cdot s_{a,i}, \quad s_{a,i} \in \{0, 1\}$$  (4.18)

with the requirement that

$$\sum_{i=0}^{k} s_{a,i} = 1$$  (4.19)

meaning that only one of the $s_{a,i}$ variables can be non-zero. This means that special order sets are used for the slack variables, which a MIP solver might use to speed up the solving process. For more information on binarisation of variables and its success, see [Bonami and Margot (2014)].

4.4.6 Quadratic objective

In the model that is defined, the objective function was linear. This means that adding two minutes on an upper bound of one arc counts just as much as spreading this out over two arcs, as long as the weights are the same. However, this last situation is often considered as better, it spreads out the adaptations over more trips or constraints. To avoid this, a non-linear objective can be implemented. However, a linear program is wanted, so it should be formulated as a linear function. There are several ways to do so.

First of all, consider the SOS-constraints as mentioned in the previous section. The contribution of a slack variable in the objective is $w_s \cdot s$. Now this slack variable is written as

$$s = \sum_{i=0}^{k} i \cdot s_i, \quad s_i \in \{0, 1\}$$  (4.20)

To implement a quadratic objective, the following term can be added to the objective function instead of the term $w_s \cdot s$:

$$\sum_{i=1}^{k} w_s^i \cdot s_i$$  (4.21)

4.4.7 Symmetry breaking

This problem includes a lot of symmetry. Having a schedule, every event can be shifted by $k \in \mathbb{Z}$ minutes, leading again to a feasible schedule, since constraints only imply time relations between two variables, not fixing just only variable. Also for the constraints that fix a certain variable to zero (absolute constraints), this does not matter. If a schedule is found that has a value different from zero for the ‘ABS’ node, it can be shifted such that this value gets zero. This symmetry makes it possible to fix one node to a certain value, thus dividing the number of possible optimal solutions by $T$.

Further, the weights of the slack variables depend on the type of constraints. Changing a trip time constraint is consider as being better than changing a frequency constraint. However, consider trip times, some have a trip time of zero minutes, while other have a trip time of eight minutes for example. It is preferred to change the largest trip time, this will cause the least amount of change in the constraint. Therefore, it might be beneficial to give the slack terms concerning trip time constraints that have a small trip time a slightly higher objective value than those that have larger trip time, to state this preference. This gives some different terms in the objective function, which might cause the MIP solver to distinguish between several solutions and not seeing them as equally good. In this model, the following weights are used for the different types of constraints:

**Type D** Upper bound: $\max\{8 - \text{triptime}, 2\}$. Lower bound: $\max\{30 - \text{triptime}, 10\}$. The larger trip times are, the cheaper it is to change them.

**Type S** Upper bound: $\max\{7 - \text{triptime}, 2\}$. Lower bound: $\max\{30 - \text{triptime}, 8\}$. It is slightly preferred to change trip times when a stop is involved. In this case, it can also correspond to a larger dwell time.
4.5. Algorithmic approach

Type A Upper bound: 4. Lower bound: 10 A slight increase in the connection time is not really a problem. In this case passengers have to wait a little bit longer, but passengers that alight at the connection station are not influenced.

Type F Upper and lower bound: 75. This is chosen to be really high. In practice this will only change if it cannot be solved by changing trip times. It is preferred not to change the frequencies, since that might trouble customers. If the solution that comes out corresponds to a large delay for some train, planners can check manually if it can also be solved by changing frequency constraints and if that might give a better solution.

Type ABS Upper and lower bound: 4. Changing an absolute constraint is not really a problem. Trains might arrive or leave the country also a bit earlier or later.

Note that frequency and absolute constraints are changed similarly. The slack variable for the upper bound is equal to that of the lower bound. Therefore, the weights are only added for the variable of the upper bound.

Having obtained a solution, another one might be just as appropriate. CPLEX may spend a considerable amount of time iterating between several solutions that are equally suitable. Therefore, it might be beneficial to give the coefficients of each term in the objective function a little perturbation to avoid having terms with the same coefficient and thus prevent the MIP solver regarding them as equally good.

4.4.8 CPLEX settings

CPLEX is very good in determining good settings that work well for the solving phase. However, it does not always understand everything about the structure of the problem and it might be better to provide this information.

After some LP relaxation is solved, it creates new problems by fixing one of the variables to an integer value of by adding extra restrictions cutting away the non-integer solution. Choosing such a variable is very important for the progress of the algorithm. Next, it chooses one of the created subproblems to continue with. Choosing such a problem also influences the progress of the algorithm. CPLEX provides several settings to make a choice on this.

4.5 Algorithmic approach

As was mentioned in the previous section, there are different methods of adding more information to the conflict that is found, in order to solve it in a good way. There are several approaches that can be taken. A few things have to be noted.

First of all, it is not known in forehand how large the conflicts are that CADANS returns. Some might be small, others might be large. Therefore, adding the same type of information to the conflicts might be risky. Although it might lead to the best solutions if a certain level of information is used, for small conflicts this even might be necessary to provide a good solution, for larger conflicts, this might lead to a huge amount of constraints, making solving hard.

Next, suppose that graph \( G_1 = (N_1, A_1) \) is a sub graph of \( G_2 = (N_2, A_2) \), meaning that \( N_1 \subseteq N_2 \) and \( A_1 \subseteq A_2 \). The objective value for the MIP-model (4.15) of some graph \( G_i \) is denoted by \( z_i \), the optimal objective value is denoted by \( z_i^* \). This leads to the following theorem:

**Theorem 4.2.** If \( G_1 \subseteq G_2 \), then \( z_1^* \leq z_2 \).

**Proof.** Note that if \( G_1 \) and \( G_2 \) contain no conflicts, one has \( z_1^* = z_2^* = 0 \) and hence the inequality holds. If \( G_2 \) contains a conflict and \( G_1 \) does not, one obviously has \( z_1^* < z_2^* \leq z_2 \) as well.

The interesting situation is when \( z_1^* > 0 \). Since \( G_1 \subseteq G_2 \), \( G_2 \) does contain at least the same conflict, possibly more. If a solution is feasible in \( G_2 \), it is feasible in \( G_1 \) as well, since \( A_1 \subseteq A_2 \), so \( G_2 \) can only contain more constraints. Since \( G_1 \) contains less constraints, a ‘cheaper’ solution might be possible there as well, which is not feasible in \( G_2 \). Therefore, \( z_1^* \leq z_2 \). 

Using this theorem, it might be beneficial to first solve the smallest graph possible, giving \( z_1^* \) as objective value. If more constraints are added, giving \( G_1 \subseteq G_i \) (for some \( i > 1 \)), a constraint can be added to the MIP model, stating that \( z_i \geq z_1^* \).

In the following, two approaches will be described that are tested in this thesis.
The first approach determines the best configuration for a conflict graph to solve a conflict. The second approach uses the theorem stated above to provide a lower bound for the MIP model of a larger conflict graph, so this follows a cut generation approach.

In order to distinguish between several graphs that involve the conflict, some notation is defined. First of all, $G_L$ is the largest graph that is considered. It is build as follows:

- Add all incoming and outgoing arcs of the nodes in the conflict.
- Add the three trips before and after the trips that are involved in the conflict for the corresponding train series.
- Add all intermediate arcs
- If single track is involved, add the corresponding arcs.
- Add all arcs that depend on the trip times involved in the graph build.

This graph can grow very large. Therefore, some sub graphs are build, they are called $G_{i,j}$ for $i \in \{0,1\}$ and $j \in \{0,...,3\}$. They are build in the following way. If $i = 1$, all incoming and outgoing arcs are added. If $j = 1$, all intermediate arcs are added. If $j = 2$, single track series are added as well, if they are involved in the conflict. If $j = 3$, three trips before and after the conflict of the involved train series are added as well, including all intermediate arcs.
4.5.1 Determine best conflict graph

As mentioned before, having a good conflict graph is crucial for the performance of the algorithm. If no extra information is added, it might imply the model gives a bad solution. The best thing to do is to take into account as much information as is possible, but this leads to larger computation times. Therefore, if too much information is added, the running times explode.

This algorithm determines a few graphs that contain the conflict that CADANS has found. First of all, a large graph is tested. If this can be solved in limited time, the solution is used to adapt constraints. If this is not possible, several subgraphs of the large graph are found. It tests whether the corresponding MIP models can be solved within a certain time limit. If this is possible, the graph that led to the highest objective value is stored as being the best. Having a higher objective value means that it provides relevant information that forbid ‘cheaper’ solutions and the solution should be valid in a larger environment of the conflict as well. Finally, the solution to the conflict graph that has led to the highest objective value is used to change the constraints and solve the conflict.

The reason to build such an algorithm is that it is not known on forehand how large the conflicts will be and how many constraints are added with the several methods of adding information. In this way, the best settings that can be found in short time are used for solving. The algorithm that is developed is shown below. Here, result=Conflex() appears twice. Conflex is the submodule of CADANS that determines a timetable or a conflict. The outcome of it is stored in ‘result’.

**Algorithm 4.3.**

<table>
<thead>
<tr>
<th>Data: PESP instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: Solution to PESP, possibly with adapted constraints</td>
</tr>
<tr>
<td>initialise etappes, constraints;</td>
</tr>
<tr>
<td>result = Conflex();</td>
</tr>
<tr>
<td>while result == conflict do</td>
</tr>
<tr>
<td>Read conflict;</td>
</tr>
<tr>
<td>Solve $G_L$ in 3 minutes;</td>
</tr>
<tr>
<td>if solution then</td>
</tr>
<tr>
<td>Adapt constraints;</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>Graph $B$ ← $G_{0,0}$;</td>
</tr>
<tr>
<td>for $i = 0$ to $1$ do</td>
</tr>
<tr>
<td>for $j = 0$ to $3$ do</td>
</tr>
<tr>
<td>Solve $G_{i,j}$ in 3 minutes;</td>
</tr>
<tr>
<td>if optimal then</td>
</tr>
<tr>
<td>Graph $B$ ← $G_{i,j}$;</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>Process solution found for $B$;</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>Rewrite etappes, constraints;</td>
</tr>
<tr>
<td>result = Conflex();</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>
4.5.2 Use a cut generation approach

In Algorithm 4.3, it might be very well possible that small graphs are chosen since the others could not be solved in short time. The best thing would be to take into account much information. The more constraints are added to the model, the better the choice will be that is made. However, proving optimality of a solution is hard in this case. Finding good solutions is often possible in limited time, but proving they are optimal is hard. Therefore, a cut generation approach is implemented.

The more constraints are added to the model, the better the choice will be that is made. However, computation times may exceed practical limits in this case. As is seen in the previous approach, several smaller instances can be solved. In this approach, much information will be added to make good choices. However, in large instances, computation times are large, but this is due to the fact that it is hard to prove optimality. During development of the algorithms, it became clear that finding good solutions can be done quite fast, especially when certain branching strategies are chosen. However, proving optimality is often very difficult. Therefore, a cut generation approach is developed. If \( G_1 \subseteq G_2 \), then \( z^*_1 \leq z^*_2 \). This can be added to the MIP-model for a graph that contains \( G_1 \). To benefit from this property, a cut generation approach is used. Small graphs are used to generate cuts, bounds on the objective value. Those small graphs are solved in limited time. Then, the MIP model of a larger graph is solved. This graph contains all incoming and outgoing arcs of the nodes in the conflict. Next, more trips of the series involved in the conflict are added. If single track is involved, all trains that use this track are added as well. Next, all intermediate arcs are added. Finally, all arcs are added that depend on trip times that are involved in the graph. This conflict graph might grow very large. Therefore, first small graphs are solved to give bounds on the objective value. This gives the algorithm that is described below.

Algorithm 4.4.

```
Data: PESP instance
Result: Solution to PESP, possibly with adapted constraints
initialise etappes, constraints;
result = Conflex();
while result == conflict do
    Read conflict;
    Try to solve \( G_L \) in 3 minutes;
    if solution then
        Adapt constraints;
    else
        lb = 1;
        for i = 0 to 1 do
            for j = 0 to 3 do
                Solve graph \( G_{i,j} \) in 3 minutes with the constraint \( z \geq lb \);
                lb ← max\{lb,bestbound\};
            end
        end
        Solve \( G_1 \) with constraint \( z \geq lb \) in 30 minutes;
        Adapt constraints with best found solution;
    end
    Rewrite etappes, constraints;
    result = Conflex();
end
```

Note that the algorithms do not differ very much. Probably Algorithm 4.4 might consume more time, since it has to solve a large graph.
Chapter 5

Results

In this chapter, the computational results are shown for several network instances. There are three instances for which the algorithms are tested. For one of the instances, several settings and tricks for speed ups are tested to see if this makes any difference for the solving process. During the development of the algorithm, some settings appeared to be useful, these are seen as default and are used for the calculation.

Programming is done in JAVA. To solve the mathematical programs, IBM ILOG CPLEX 12.6.1 is used, with up to 8 parallel threads. Computations are done on a 64-bit computer with an Intel(R) Core(TM) i7-4700MQ 2.40GHz processor with 8.00 GB of RAM installed, operating under Windows 8.1.

First of all, the different instances for testing are described. After that, some results of the calculations are shown. Note that it is not very informing to show a full timetable, this would be a long list of departure times providing no useful information. Instead, it is shown how many times a conflict is found and how many constraints are changed in order to obtain a feasibility.

5.1 Description of test instances

In this section, the several instances that are used to test the algorithms are described. Further, they are given a name that is used to refer to them in this chapter.

5.1.1 Hdr30: Single track network Hdr-Sgn

This network is very small, containing only six train-path points. There is one intercity train using this track, which is driving two times per hour, in a 30/30 pattern in both directions. There is one sprinter train using this track, driving only once per hour, also in both directions. For CADANS this means there are six trains using this track. In total 30 departure events have to be found. The number of constraints that hold for this network amount up to 184. The network and the constraints that come into play are described in Appendix E.

5.1.2 NL13: Complete Dutch network 2013-2018, adapted by planners

This instance is a model that includes a full timetable for the whole of the Netherlands, including trains of other companies than Netherlands Railways. This model was used to design the basic railway timetable for 2013-2018. In 2018, NS wants to operate a new timetable, designed from scratch. This is done in order to drive with six intercity trains and six sprinter trains on the A2-corridor, which is the line Eindhoven-Utrecht-Amsterdam. This plan is called ‘Programma Hoogfrequent Spoor’ (PHS). The model that is used here was used to test to what extend it would be possible to incorporate this plan in the current timetable. Planners have adapted some trip times in this schedule already, in order to solve some conflicts. However, it is still infeasible.

In total, it includes 448 trains and 9084 departure events have to be planned. The number of constraints amount up to 80913.
5.1.3 **NL13new**: Complete Dutch network 2013-2018, no changes

In order to test how the algorithm performs on a network that is used to design a timetable from scratch, all trip time changes that were made in the NL13 instance are made undone. This is a ‘new’ instance to design a timetable for 2013, without any changes made by planners. They have changed a lot of trip times, for almost each train series some trip times are changed, if not many of them. This model therefore might include a lot of conflicts.

Basically, the model is the same as the previous one. However, a few constraints are added for the super series. Normally, they are only synchronized at the first station it starts as a super series. Planners take care the trip times are the same. Here, more frequency constraints are stated that take care of the synchronisation at the other shared stations as well. Further, sometimes planners add some extra constraints, especially on single track. Often this is done to make some changes based on trip time changes they have made or in order to move some lines in a wished position. This is made undone as well. In total the number of constraints in this model amount up to 75157.

The number of trains is again 448 and the number of departure events 9084.

5.2 Computational results

There are different configurations to be tested for solving the MIP. However, it is not good to take a single MIP problem to test it on, since that might give biased results. Some settings might work well for one MIP, while making it worse for other problems. Further, some tricks might work well on large instances, while having a bad effect on small instances and vice versa. Further, the difficulty to solve a MIP largely depends on the size of the problem. If it is small, tricks will have no noticeable effect. If it is large, it will take a long time anyway and tricks will not have that much influence. Somewhere between small and very large is a ‘region’ for which the tricks might work. Therefore, a full run of an algorithm is performed to test the different settings, to see how much it influences the total time it takes to solve an instance. In this thesis, tests are done using Algorithm 4.3 on the NL13 instance. This algorithm has the best potential to be the fastest. Concerning the instance, it is known that only few conflicts occur.

As a start, settings are chosen that during development appeared to be promising. Those are the following: Provide extra cuts on the \(p\)-values within the cycles of the cycle basis in order to prune branches of the search tree. Next, make the coefficients of the slack variables corresponding to trip time constraints dependent on the trip time, i.e. make it ‘cheaper’ to change a trip time constraint that corresponds to a large trip time than that corresponding to a small trip time. In order to further differentiate between solutions, a quadratic objective is implemented together with SOS-constraints for the slack variables. To avoid some unnecessary branching, the departure event times are not required to be integer. Also, branching priorities are given to the \(p\)-values as is described in Section 4.4.4, thereby fixing the node with the highest degree to zero. Finally, if train series arcs are added, three trips before and after the trips in the conflict are added.

5.2.1 Results on Hdr30

This instance is that small that solving is done very fast. This means that differences between the algorithms or settings of CPLEX will not be noticeable. Therefore, only the results of the computation will be stated here.

The first (and only) conflict that is found involves 25 nodes and 38 arcs. This corresponds to both intercity lines and the sprinter line in one direction. However, this network is a single track network, therefore the other trains on the track have to be taken into account in order to make a good solution. If this is not done, the result is that a few times after each other almost the same conflict has to be solved.

Solving the network is done within a second. In total, less than three seconds were needed to run the algorithm, including running CADANS twice, first to find a conflict, after that to build a timetable. The changes that are made are shown in Table 5.1.

The last column shows the original trip time and the increase or decrease. Note that all trip times that are changed in this case, are decreased by one minute. This means trains have to drive faster than scheduled. Probably this is not always possible, but this depends on the bounds on the slack variables, which planners might provide.
5.2. COMPUTATIONAL RESULTS

### Results on Hdr30

<table>
<thead>
<tr>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>468</td>
<td>T2</td>
<td>Sgn Ana</td>
<td>Trip 5 – 1</td>
</tr>
<tr>
<td>469</td>
<td>H1</td>
<td>Hdr Hdrz</td>
<td>Trip 4 – 1</td>
</tr>
<tr>
<td>513</td>
<td>T1</td>
<td>Sgn Ana</td>
<td>Trip 5 – 1</td>
</tr>
</tbody>
</table>

Table 5.1: Results on Hdr30

### Results on NL13

#### Using algorithm 4.3

In this section, results are shown on testing several settings to solve the MIP-models that are built by the algorithm. First of all, the instance is solved with default settings. Next, several settings are changed. The computational results are shown in Table 5.2. The last column shows how much time is used by CPLEX for solving the models, including the percentage it is of the total running time of the algorithm. For detailed results on the calculations, see Appendix C.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Conflicts</th>
<th>Arcs</th>
<th>Dependent arcs</th>
<th>Total time (sec)</th>
<th>Cplex time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>2</td>
<td>9</td>
<td>37</td>
<td>240</td>
<td>197 (82 %)</td>
</tr>
<tr>
<td>Linear objective</td>
<td>2</td>
<td>7</td>
<td>25</td>
<td>231</td>
<td>189 (82 %)</td>
</tr>
<tr>
<td>No SOS</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>456</td>
<td>287 (63 %)</td>
</tr>
<tr>
<td>Do not limit $p$</td>
<td>2</td>
<td>6</td>
<td>25</td>
<td>231</td>
<td>173 (75 %)</td>
</tr>
<tr>
<td>$p$ equal priority</td>
<td>2</td>
<td>7</td>
<td>31</td>
<td>218</td>
<td>67 (31 %)</td>
</tr>
<tr>
<td>No priorities</td>
<td>2</td>
<td>6</td>
<td>22</td>
<td>231</td>
<td>81 (35 %)</td>
</tr>
<tr>
<td>Use perturbation</td>
<td>2</td>
<td>8</td>
<td>40</td>
<td>262</td>
<td>218 (83 %)</td>
</tr>
<tr>
<td>Integer event times</td>
<td>2</td>
<td>9</td>
<td>37</td>
<td>280</td>
<td>237 (85 %)</td>
</tr>
</tbody>
</table>

Table 5.2: Summary of results on NL13, algorithm 4.3

This instance appeared to have two conflicts that had to be resolved. If only few information from the surroundings of the conflict is taken into account, the algorithm might have to do several attempts to solve a conflict, since one solution might imply another or it does not make a good solution globally. However, in this instance that was not the case. All conflicts were solved in one go. Further, the first and the second conflict involved different train series and did not influence each other.

In Appendix C, the different solutions are shown that were obtained. They all involve the same train series, but there are several different solutions available that are equal according to the objective function. Depending on the settings, CPLEX finds different solutions. For planners, it would be interesting to see (if possible) all solutions, so they can choose from different solution, what they believe is the best.

Concerning the calculation times, it can be seen that a lot of time is spent by other parts of the algorithm than solving the MIP’s. On average, 72.1% of the running time is used by CPLEX. Finding graphs can be done very rapidly and finding a tree and a cycle basis is a problem neither. The main part of the remaining time is consumed by CADANS finding a timetable. During the runs, conflicts were found very soon, but proving that a timetable exists by finding one sometimes took at least a minute.

For the different settings, two are really advantageous for the models solved here. Giving all $p$-variables equal branching priority reduced the solving time by 66% when compared to default settings. Skipping the branching priorities at all, reduced solving time by 59%, compared to default settings. An interesting thing to note here is that using the branching priorities leads to having a good solution rapidly. However, proving optimality is hard. If all $p$-variables are given equal branching priority, or if they get no priority at all, proving optimality is done much faster.

Next to this, implementing a quadratic objective function did not really make a difference here. Computation times are basically the same. However, putting a SOS-relation for the slack variables, i.e. adding the constraint

$$s = \sum_{i=0}^{k} i \cdot x_i, \quad \sum_{i=0}^{k} x_i = 1, \quad x_i \in \{0, 1\}$$  \hspace{1cm} (5.1)
appeared to be very beneficial. In this case, the objective is linear as well, since implementing a quadratic objective function is not easy if one does not want to write it as a sum of other (binary) variables. If the SOS-formulation is left out, computation times increased by almost 46%.

Using algorithm 4.4

In Table 5.3, the results are shown on the NL13 instance. The algorithm is run with default settings and with giving each $p$-variable equal branching priority.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Conflicts</th>
<th>Arcs</th>
<th>Dependent arcs</th>
<th>Total time (sec)</th>
<th>Cplex time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default $p$ equal priority</td>
<td>2</td>
<td>9</td>
<td>37</td>
<td>237</td>
<td>195 (82%)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>31</td>
<td>214</td>
<td>64 (30%)</td>
</tr>
</tbody>
</table>

Table 5.3: Summary of results on NL13, algorithm 4.4

Note that the results are basically equal to the results of Algorithm 4.3. Computation times only differ by seconds, so no significant differences are shown. The conflict graph could be solved in one go, without the need to generate cuts to prove optimality. Therefore, computation times are more or less equal.

5.2.3 Results on NL13new

This instance might contain a lot of conflicts, that is not known in advance. Also, the number of conflicts it finds might depend on the way previously found conflicts are resolved.

Using algorithm 4.3

This algorithm is tested using default settings. Further, it is tested using equal branching priorities for the $p$-variables, since that appeared to work better for the NL13 instance. The results of the calculations are shown in Table 5.4. For detailed results, see Appendix C.1.2.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Conflicts</th>
<th>Arcs</th>
<th>Dependent arcs</th>
<th>Total time (sec)</th>
<th>Cplex time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default $p$ equal priority</td>
<td>18</td>
<td>71</td>
<td>436</td>
<td>10046</td>
<td>9847 (98%)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>80</td>
<td>490</td>
<td>8106</td>
<td>7910 (98%)</td>
</tr>
</tbody>
</table>

Table 5.4: Summary of results on NL13new, algorithm 4.3

Those results show that for the initial plans of the full Dutch railway network, conflicts can be repaired and a timetable can be found within three hours of computation time.

Note that again using equal priorities for the $p$ variables showed a reduction in computation time, this time by almost 20%. It does show however that it is not as profitable as it was for the NL13 instance. Especially for large conflict graphs, the branching priorities appeared to be beneficial.

Some very interesting things can be noted in the solutions, see Appendix C.1.2 First of all, all conflicts are repaired by only changing trip times. Note that this is often not the only solution that is possible. It might also be an option to change frequency constraints. Next, by the solution that the algorithm gave using default settings, conflicts 13-16 all involve the same train series. This is a typical example of ‘uitbuigen’. The conflicts occur on the line between Zwolle and Groningen. Here, intercity trains have to be delayed in order not to get too close to the sprinter trains. Note further that this problem is solved in four conflicts. Probably it could have been possible to solve the whole line in one go, thus saving a lot of time. Especially conflict 16 consumed a lot of time (17 minutes).

Using algorithm 4.4

Using this algorithm in the way it is defined led to a computation time of over 24 hours for the NL13new instance. Changing the algorithm a little bit by solving it for $G_{0,3}$ instead of $G_L$, i.e.,
5.3. SUMMARY OF THE FINDINGS

<table>
<thead>
<tr>
<th>Setting</th>
<th>Conflicts</th>
<th>Arcs</th>
<th>Dependent arcs</th>
<th>Total time (sec)</th>
<th>Cplex time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>43</td>
<td>109</td>
<td>425</td>
<td>1452</td>
<td>1068 (74 %)</td>
</tr>
</tbody>
</table>

Table 5.5: Summary of results on NL13new, algorithm 4.4

leave out the adding of incoming and outgoing arcs in the graph, makes it possible to solve very fast. In Table 5.5 it is shown that the instance can be solved within 25 minutes.

This result shows that the algorithm performs very fast, when $G_{0,3}$ is used instead of $G_L$. However, investigation of the results as they are shown in Section C.2 shows that cycling occurs. A few times when a change is made, it has to be made undone later on. Further, conflicts 3-16 and 25 all involve series 801 and 802. Also, conflicts 30-37 all involve series 260, 500 and 501. Although those conflicts separately are all solved very fast, it means repeatedly running CADANS, consuming quite some time, and for many iterations no good solutions are made that solve the conflict for good. This leads to changing more constraints than is strictly necessary. When the results are compared to that of the other algorithm, 53% more constraints are changed. Therefore, although this algorithm is fast, it changes more constraints than is necessary. It can be used for a quick check to resolve some conflicts, but for a better solutions, the other algorithm perform better concerning the quality of the solution.

5.3 Summary of the findings

As is shown in this chapter, timetables can be calculated within a few hours of computation time. If the PESP graph allows for the existence of a feasible timetable, it can be found very fast. However, if there are infeasibilities to resolve, it takes some more time, although finding a timetable within only three hours is just fine. It gives planners a good timetable to start with if a new timetable is to be designed.

Algorithm 4.4 appeared to work fine for instances that do not contain many conflicts. It has the advantage that it always takes into account a lot of information, leading to high quality solutions. However, if there are a lot of conflicts, algorithm 4.3 comes up with good solutions faster.
Chapter 6

Conclusion

As was stated in Chapter 3, the solver CADANS has been in use at NS and ProRail for over 20 years. In those years, many things have changed in planning processes, but CADANS remained the same. CADANS has some shortcomings, e.g., if the constraints prevent a feasible solution to exist, a conflict is returned, which is hard to understand. In this case, no timetable is available to use for the development of a timetable, even not for the parts in the network that have no conflicts involved. It would be interesting to have a timetable that satisfies all safety constraints, but might some of the others if this is necessary to have a timetable.

In this thesis, an answer is given to the following research question, as it is formulated in Chapter 1:

How can a feasible railway timetable be found with minimal deviations from the original constraints in the case that initially no feasible timetable exists?

The methodology that is used by CADANS is constraint programming. When the set of constraints is constructed in such a way that a feasible timetable exists, this timetable will be found in a short time, in the order of minutes. When no feasible timetable exists, CADANS often detects a conflict quickly and returns a minimal set of conflicting constraints. If this set of conflicting constraints (in short: conflict) is solved, this does still not guarantee that a feasible timetable exists. It can occur that many more conflicts exist, since the solution that is used to solve the conflict might imply another conflict. However, at least one conflict is repaired.

In this thesis, two algorithms are proposed as an extension to the CADANS solver. The algorithms changes the original set of constraints in such a way that a feasible timetable exists. The changes are as few and as small as possible. The proposed algorithms repeatedly run the Conflex module from CADANS. When a conflict is found, a solution is found and the constraints are adapted. If at some time instance during the run of the algorithm no conflict is found anymore, CADANS returns a timetable. This returned timetable then satisfies all safety constraints. Some trip time constraints or other requirements may be violated, however, those violations are minimized.

In the end, the proposed algorithms return a feasible timetable. Conflicts are solved iteratively and although some cycling occurs, using the penalties that are used in this method, a good solution to the conflict comes out in the end. There is no theoretical guarantee for the convergence of the proposed methods, since it might be possible that the algorithm does not find a solution that avoids cycling. However, in practice the algorithm results in a feasible timetable.

The timetabling problem is stated as a Periodic Event Scheduling Problem. It consists of constraints that state a relation in time between two events. The difference in time between the events should be within a certain periodic interval. In case conflicts exist, those bounds are put too tight for a feasible solution to exist. Solving a conflict is done using a Mixed Integer Program. The MIP model adds slack variables to the bounds of the constraints that are involved in the conflict, i.e., the bounds of those constraints are relaxed. Concerning trip times, these are not relaxed, since they can not be made flexible in the model as it is used currently. Many constraints depend on the trip times, so the corresponding slack variables are integrated into the bounds of those constraints as well. Next, a weighted sum of the slack variables is minimised, thus providing a minimal deviation from the original constraints, while guaranteeing that the conflict now is made feasible.
The found solution might imply other conflicts or cause the conflict to be postponed. Therefore, it is shown to be important to incorporate more ‘information’ from the surroundings of the conflict into the constraint graph, i.e., add extra constraints. This leads to better choices to solve the problem. Both of the algorithms rely on the fact that if more constraints are added to the conflict graph, the optimal objective value will never decrease.

In the results of the calculations it is shown that Algorithm 4.3 could solve the conflicts in the NL13new instance within three hours. Most conflicts are solved in a good way, the conflicts are not postponed. This is due to the fact that the constraint graph is chosen to solve the conflict, that has lead to the highest objective value in the corresponding MIP-model. A slightly changed version of Algorithm 4.4 is able to solve the same instance within one hour and most conflicts are solved in only a few seconds. However, using the algorithm the way it is stated originally has lead to a time out since the instance could not be solved within 24 hours. It took into account a lot of information, thus having a high probability of making good choices, but this corresponds to a large computation time. If another conflict graph is chosen to solve the conflicts, this has lead to a solving time of less than one hour. Using less information, leads quickly to a solution. On the other hand, many solutions only postponed the conflict and changes were made undone in a next iteration. This means that the solution that is found in the end might violate more constraints than actually is necessary. No proof of convergence to a good solution can be given here, although all calculations that are done in the end resulted in a feasible timetable.

Designing a new timetable from scratch is done in 2007 for the last time. The next plans to design a totally new timetable are for 2018. This means designing a new timetable is done rarely. Therefore, computation times do not matter that much. However, every year in December, small changes are made to the timetable. Sometimes an extra train has to be added on a track. This often gives rise to small conflicts. Some constraints have to be adapted in order to make everything fit on the track. By using the proposed approach, it is possible to find feasible schedules within only a few hours. If only small changes to an existing timetable are required, they are found within minutes or even shorter. For a strategic study, finding minimal adaptations requires a few hours of computation time.

The proposed algorithms provide a functional extension to the CADANS solver and provide better information than the options-module. It solves at least one of the shortcomings that CADANS currently has. From now on, planners at NS can create a feasible timetable for the whole country, even if the constraints are such that initially no feasible timetable could exist. This might help them in the development of a new timetable. If the violated/adapted constraints are shown to the planners, they can decide whether the proposed solution is acceptable or if they want to change it.
Chapter 7

Discussion and recommendations

CADANS is a solver that performs well under certain circumstances. The last years circumstances have changed, and planners at NS and ProRail experience the shortcomings of CADANS more than its benefits. This is one of the reasons that CADANS is not used often any more. One of the shortcomings of DONS and CADANS that planners experience is that planning is done in full minutes, a cycle time of 60 (minutes) is used. However, it would be better to use a finer time grid, especially since nowadays many trains are on the track. Many people would like to use a cycle time of 600 (tenths of minutes). Currently, CADANS is only able to deal with a time grid of 60 (minutes) or 120 (half minutes). Another shortcoming that CADANS has is that it heavily rests on the assumption that trip times are fixed. This allows for a reduction of the constraint graph, but cuts away a lot of flexibility. Next to this, the conflicts that CADANS returns are often hard to understand. This is not a fault of CADANS, it is merely due to the complexity of the constraint graph. Another circumstance that has changed is that there are a lot more trains on the network when compared to 20 years ago. This makes it hard to fit everything on the tracks. Another reason for not using CADANS is that an extension to DONS has been implemented lately, making it possible to draw time space diagrams by hand, thus making the planning process easier.

In this thesis an extension is provided to overcome to shortcoming of only getting conflicts from CADANS. This extension resolves the conflicts in an automatic manner. However, planning is still done in full minutes. Further, trip times are still assumed to be fixed, although this method might change some trip times around the conflict, to resolve those conflicts. In some sense, it solves this shortcoming as well. However, if the time grid would be changed, one most probably does not want to assume fixed trip times any more. Fixing trip times is not a good situation when a very detailed time grid is chosen, changing it by six seconds for example does not make any difference.

The topics mentioned above show that CADANS and this model are open to improvements. If one lets go of the assumption of fixed trip times, the number of events for which a time is to be determined basically doubles. On the other hand, many of the constraints get much more intuitive. Also, this model would become much easier and probably many conflicts would not pop up if trip times are allowed to be flexible, since they could be solved by only a slight change in the trip times. It would be interesting to investigate how well the methodology of CADANS deals with a network that depends on arrival events as well. In Kroon and Peeters (2003), a method is shown by which it is possible to incorporate flexible trip times. This can either be done by adding extra constraints, or by inserting virtual dummy stations and giving a train the possibility to ‘stop’ there for a limited time, thus giving it a little delay.

Next to this, it would be interesting to see how well CADANS can deal with a finer time grid. Currently, every node in the graph gets a set of possible values: \{0, 1, \ldots, 59\} assigned, which is restricted during the run of the algorithm. If the cycle time $T$ is changed to $T = 600$, i.e. use a time grid that uses periods of tenths of minutes, each node gets the set \{0, 1, \ldots, 599\} of possible values, so the algorithm has much more choice. This makes it more difficult to restrict the size of this value set during the run of the algorithm, probably increasing the running time of the algorithm considerably.

Next to extending CADANS to a finer time grid, the MIP model that is proposed in this thesis can be extended to this grid as well, even if CADANS is not changed. If this is done, it is possible to determine if constraints have to be violated only slightly, i.e., by a few seconds, or that they have to be violated by a few minutes. In order to do so, the headway constraints would have to be
CHAPTER 7. DISCUSSION AND RECOMMENDATIONS

recalculated as well, since everything is in full minutes right now. Hence, the final solution for the deviations from the original constraints would most probably also be in full minutes. Probably in doing this change in the model, it provides more insight in the conflict.

The algorithm that is proposed in this thesis does solve Mixed Integer Programs with a lot of big M constraints. Although clever branching methods might be chosen, solving might still consume a lot of time, especially to prove optimality in the minimisation problems. It would be interesting to find some tunings that speed up the process of solving.

Furthermore, the algorithms that are proposed can be improved as well. In some iterations, a lot of time is lost when no improvements are found to a MIP during some time. If this can be detected earlier, the calculation might be stopped and another conflict graph might be chosen to continue with.

The method proposed in this thesis is in fact a heuristic method, no proof of optimality or convergence can be given. The conflicts are solved only locally. It might happen that the algorithms get into a cycle, changes that are made in some iterations have to be made undone later on. In the current model, this is given a higher penalty which appears to lead to a good solution in the end, to get out of the cycling. However, there is no guarantee that this will happen. Another way to avoid this cycling is by using a TABU-list, i.e., constraints that are adapted in some iterations can not be changed in the $n$ iterations after that.

Furthermore, the best solution, and probably the optimal one, will come out if the full constraint graph is taken into account. However, it will most probably consume a huge amount of time to solve such problem to optimality. The more information is taken into account, the more time it costs to solve the MIP models.

Next to the models and algorithms that are used here, it would be interesting to use a constraint programming approach to solve this problem, integrating it to the constraint programming of CADANS. In this way, both the timetabling and solving of conflicts are integrated. Also, it would be interesting to find a heuristic method that solves the conflicts, not using the MIP solver. Although this provides good solutions for small problems, as soon as the problem size grows, the solving time grows considerably as well. Especially for those cases, a heuristic would be interesting and probably beneficial.

If a new timetable is designed, initially many conflicts may occur in the network. Probably the best solution would come out if the full network would be solved using the MIP method proposed in this thesis. However, this will take a very long time before a solution will be found, which is not desired. For now, the iterative procedure works just fine, but it would be interesting to find a way to solve the whole network in one go. As mentioned before, probably this could be done using a heuristic approach.

As is clearly seen in this thesis, all constraints depend solely on the departure events. This is quite a limitation. It more or less prevents the possibility to introduce flexible trip times, although Kroon and Peeters (2003) provide constraints that can be stated such that flexible trip times would be possible, although it is easier to insert dummy stations in the DONS system where trains have the possibility to virtually stop for some time. It would be interesting to investigate how well the CADANS algorithm can deal with a network that also includes the arrival events. In this case, the number of events double, but many constraints get much easier and it allows for more possibilities in modelling.

In this thesis, some results are shown for several settings to help CPLEX in solving the Mixed Integer Programs. It would be interesting to do more research on finding better settings to solve the programs. Also, in the algorithms things are to be improved, especially in the JAVA code that is used. Some functions might be implemented more efficiently which might lead to some savings in time.

In the presented solutions, only trip times are changed. This does not have to be the only solution, it depends on the weight in the objective function which solution is chosen. However, violating frequency constraints is not a desired solution, since this makes things fuzzy for customers. A timetable is easier to remember if it has the same patterns, not only an hourly pattern, but if possible a 30-minute pattern. Further research can be done on a method to provide multiple solutions to solve the conflicts.

In practice, a timetable is as much symmetric as possible. In the algorithms that are proposed, this is not taken into account. It can easily be added to the model, but it might imply that another constraint graph has to be used, since all trains of a certain series have to be incorporated. Further, if a trip time is changed for the first train of a series, it might be good to change it for the second
train as well. This would mean that the slack variable for some trip of a train should be the same
for the other train of the series making the same trip. All of those properties can be added to the
model that solves the conflicts. The main problem in using this approach is that the size of the
graph that is used to solve the conflict might get very large.
Appendix A

Abbreviations train path points

On the following page, a table is shown containing the abbreviations for the train-path points that are used in this thesis.
### APPENDIX A. ABBREVIATIONS TRAIN PATH POINTS

<table>
<thead>
<tr>
<th>Ac</th>
<th>Abcoude</th>
<th>Hgv</th>
<th>Hoogeveen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ah</td>
<td>Arnhem</td>
<td>Hgwbr</td>
<td>Gouwe brug</td>
</tr>
<tr>
<td>Amf</td>
<td>Amersfoort</td>
<td>Hlm</td>
<td>Haarlem</td>
</tr>
<tr>
<td>Aml</td>
<td>Almelo</td>
<td>Hrn</td>
<td>Haren</td>
</tr>
<tr>
<td>Amr</td>
<td>Alkmaar</td>
<td>Hsbdg</td>
<td>Belgische grens</td>
</tr>
<tr>
<td>Ana</td>
<td>Anna Paulowna</td>
<td>Hsbdgo</td>
<td>Breda aansluiting zuid</td>
</tr>
<tr>
<td>Apd</td>
<td>Apeldoorn</td>
<td>Hszha</td>
<td>Zevenbergschen Hoek</td>
</tr>
<tr>
<td>Asl</td>
<td>Amsterdam Centraal</td>
<td>Ht</td>
<td>'s Hertogenbosch</td>
</tr>
<tr>
<td>Asdm</td>
<td>Amsterdam Muiderpoort</td>
<td>Hvs</td>
<td>Hilversum</td>
</tr>
<tr>
<td>Asn</td>
<td>Assen</td>
<td>Hwk</td>
<td>Harderwijk</td>
</tr>
<tr>
<td>Asnz</td>
<td>Assen Zuid</td>
<td>Hwko</td>
<td>Koekange</td>
</tr>
<tr>
<td>Asra</td>
<td>Amsterdam Riekerpolder aansluiting west</td>
<td>Kg</td>
<td>Koegasbrug</td>
</tr>
<tr>
<td>Ass</td>
<td>Amsterdam Sloterdijk</td>
<td>Kgs</td>
<td>Koegasbrug</td>
</tr>
<tr>
<td>Atn</td>
<td>Aalten</td>
<td>Krg</td>
<td>Kruiningen-Yerseke</td>
</tr>
<tr>
<td>Atwg</td>
<td>Antwerpen Groenenlaan</td>
<td>Lc</td>
<td>Lochem</td>
</tr>
<tr>
<td>Bhx</td>
<td>Boven Hardinxveld</td>
<td>Ldmw</td>
<td>Leerdam West</td>
</tr>
<tr>
<td>Bke</td>
<td>Berkel-Enschot</td>
<td>Lr</td>
<td>Laren-Almen</td>
</tr>
<tr>
<td>Bkl</td>
<td>Breukelen</td>
<td>Mk</td>
<td>Markelo</td>
</tr>
<tr>
<td>Bl</td>
<td>Beilen</td>
<td>Mkbr</td>
<td>Zederik-Merwedekanaal Arkel</td>
</tr>
<tr>
<td>Bmr</td>
<td>Boxmeer</td>
<td>Mp</td>
<td>Meppel</td>
</tr>
<tr>
<td>Bp</td>
<td>Buitenpost</td>
<td>Mpa</td>
<td>Meppel aansluiting</td>
</tr>
<tr>
<td>Brecht</td>
<td>Brecht</td>
<td>Mt</td>
<td>Maastricht</td>
</tr>
<tr>
<td>Ck</td>
<td>Cuijk</td>
<td>Nm</td>
<td>Nijmegen</td>
</tr>
<tr>
<td>Ddr</td>
<td>Dordrecht</td>
<td>Odz</td>
<td>Oldenzaal</td>
</tr>
<tr>
<td>Ddv</td>
<td>Dedemsvaart</td>
<td>Onz</td>
<td>Omnen zuid</td>
</tr>
<tr>
<td>Did</td>
<td>Didam</td>
<td>Ost</td>
<td>Olst</td>
</tr>
<tr>
<td>Dtc</td>
<td>Doetinchem</td>
<td>Rb</td>
<td>Rijland-Bath</td>
</tr>
<tr>
<td>Dtc</td>
<td>Doetinchem De Huet</td>
<td>Rlba</td>
<td>Rotterdam Lombardijen aansluiting zuid</td>
</tr>
<tr>
<td>Dv</td>
<td>Deventer</td>
<td>Rm</td>
<td>Roermond</td>
</tr>
<tr>
<td>Dvn</td>
<td>Duiven</td>
<td>Rd</td>
<td>Roosendaal</td>
</tr>
<tr>
<td>Ed</td>
<td>Ede Wageningen</td>
<td>Rta</td>
<td>Rotterdam Alexander</td>
</tr>
<tr>
<td>Ehv</td>
<td>Eindhoven</td>
<td>Rtd</td>
<td>Rotterdam</td>
</tr>
<tr>
<td>Em</td>
<td>Emmerich (D)</td>
<td>Sgn</td>
<td>Schagen</td>
</tr>
<tr>
<td>Es</td>
<td>Enschede</td>
<td>Shl</td>
<td>Schiphol</td>
</tr>
<tr>
<td>Etn</td>
<td>Etten-Leur</td>
<td>Tbg</td>
<td>Terborg</td>
</tr>
<tr>
<td>Gd</td>
<td>Gouda</td>
<td>Ut</td>
<td>Utrecht</td>
</tr>
<tr>
<td>Gdr</td>
<td>Gaanderen</td>
<td>Utg</td>
<td>Uitgeest</td>
</tr>
<tr>
<td>Gk</td>
<td>Grijpskerk</td>
<td>Vama</td>
<td>Vam aansluiting (close to Wijster)</td>
</tr>
<tr>
<td>Gn</td>
<td>Groningen</td>
<td>Vga</td>
<td>Vught aansluiting</td>
</tr>
<tr>
<td>Gnl</td>
<td>Groningen losplaats</td>
<td>Vl</td>
<td>Venlo</td>
</tr>
<tr>
<td>Go</td>
<td>Goor</td>
<td>Vlb</td>
<td>Vierlingsbeek</td>
</tr>
<tr>
<td>Gr</td>
<td>Gorinchem</td>
<td>Vsv</td>
<td>Varsseveld</td>
</tr>
<tr>
<td>Gvc</td>
<td>Den Haag Centraal</td>
<td>Wgm</td>
<td>Watergraafsmeer</td>
</tr>
<tr>
<td>Hdp1</td>
<td>Hoendiep (bridge)</td>
<td>Wl</td>
<td>Wehl</td>
</tr>
<tr>
<td>Hdr</td>
<td>Den Helder</td>
<td>Wspl</td>
<td>Westelijke splitsing (HSL close to Rtd)</td>
</tr>
<tr>
<td>Hdrz</td>
<td>Den Helder Zuid</td>
<td>Ww</td>
<td>Winterswijk</td>
</tr>
<tr>
<td>Hea</td>
<td>Herfje Aansluiting</td>
<td>Zd</td>
<td>Zaandam</td>
</tr>
<tr>
<td>Hfdm</td>
<td>Hoofddorp midden</td>
<td>Zh</td>
<td>Zuidhorn</td>
</tr>
<tr>
<td>Hgl</td>
<td>Hengelo</td>
<td>Zl</td>
<td>Zwolle</td>
</tr>
<tr>
<td>Hglgp</td>
<td>Hengelo Gezondheidspark</td>
<td>Zv</td>
<td>Zevenaar</td>
</tr>
<tr>
<td>Hglo</td>
<td>Hengelo Oost</td>
<td>Zvbtwa</td>
<td>Zevenaar oost aansluiting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zww</td>
<td>Zwaagwesteinde</td>
</tr>
</tbody>
</table>

Table A.1: Abbreviations for nodes on the network
Appendix B

Reducing a graph

This is an example of how a graph can be reduced. In Figure B.1, the original graph is shown. In Figures B.2 and B.3, the result is shown of removing the arrival nodes. This is what the DONS system does.

Figure B.1: Original graph

Figure B.2: Removing three arrival nodes

Figure B.3: Removing last arrival nodes

When the arrival nodes are removed, the graph is sent to CADANS/Conflex. What happens in the preprocessing phase is shown in the next graphs.
First of all, arcs with \( l = u \) are removed. That is for one arc in this case, so the node at the head of the arc is removed and the arcs are merged. The result is shown in Figure B.4.

![Figure B.4: Removing an arc with \( l = u \)](image)

Now the two parallel arcs can be taken together, resulting in the graph of Figure B.5.

![Figure B.5: Merging two arcs into one](image)

Again, the arc with \( l = u \) can be merged, leading to the graph of Figure B.6. This can be simplified by taking the parallel arcs together, leading to the final graph in Figure B.7.

![Figure B.6: Removing an arc with \( l = u \)](image)
A solution can now be calculated, based on the graph of Figure B.7. Calculating back leads to the possible solution in Figure B.8. The event times are shown in red in the nodes. The process times are shown in blue next to the arcs.
Appendix C

Detailed results

In this appendix, detailed calculation results will be shown. For each setting, several things are shown. First of all, characteristics of each conflict are mentioned. Further, solution times are mentioned for the conflict.

C.1 Results on finding best conflict graph

In this section, the results on the algorithm that tries to find the best conflict graph are mentioned. First of all, the characteristics of each conflict are shown. In the table, four lines are given, stating the following information on the number of nodes and constraints:

1. Size of the conflict that CADANS returns.
2. Size of this conflict if all dependent constraints are added.
3. Size of the largest conflict graph that is taken into consideration ($G_{1,3}$ is it is called in Section 4.5).
4. Size of the conflict graph that is used for the final solving in the algorithm.

Next to this, the time (in seconds) is mentioned that was needed to solve this problem. This time is only used by CPLEX. The algorithm also spends time on the CADANS solver to determine a conflict or a timetable. Especially finding a timetable costs quite some time.

At the bottom of each table, several things are mentioned as well. First of all, the total time that the algorithm has run is mentioned, with the time and percentage of this time that is used by CPLEX. The line below shows the number of constraints that are relaxed, like trip time constraints, frequency constraints and absolute constraints. Finally, the number of constraints that changed because of the trip times changing is mentioned.

Below, results are given for the instances NL13 and NL13new.

C.1.1 Result on NL13

This section shows the result for the network that contained every train in the Netherlands and in which planners have adapted a few things already. It contains only two conflicts, hence solving is done fast. For several settings in solving the models, the results are shown. For each settings, two tables are shown. The first contains information on the conflicts that are found.
APPENDIX C. DETAILED RESULTS

Default settings

Using default settings, the following results are obtained:

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 197/240 (82.1%)
Number of adapted constraints: 9
Number of dependent constraints changed: 37

The following table summarizes the changes that are made. Here, in the column train, three things are mentioned. First of all, the first three digits indicate the train series number. The next letter indicates its direction, forward (H) or backward (T). Finally, the last digit states which train of the series it is, the first or the second.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>815</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 6 + 2</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 6 + 2</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 6 + 1</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>Odz</td>
<td>Hglo</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>Hglo</td>
<td>Hgl</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>Mk</td>
<td>Lc</td>
<td>Triptime 6 + 3</td>
</tr>
</tbody>
</table>

Use a linear objective

Using a linear objective, the computation seems to be a little bit faster. However, now less constraints are changed, the changes are not spread out over more constraints.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 189/231 (81.8%)
Number of adapted constraints: 7
Number of dependent constraints changed: 25

The disadvantage of this method is that it does not spread out the changes over more constraints if this is possible. Now only seven constraints are changed, while it were nine constraints with default settings.
C.1. RESULTS ON FINDING BEST CONFLICT GRAPH

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td>1</td>
<td>H2</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>1</td>
<td>H2</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>1</td>
<td>T1</td>
<td>Odz</td>
<td>Hglo</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>1</td>
<td>T1</td>
<td>Ddn</td>
<td>Go</td>
<td>Triptime 8 + 1</td>
</tr>
<tr>
<td>1</td>
<td>T1</td>
<td>Mk</td>
<td>Lc</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>2</td>
<td>815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Mp</td>
<td>Triptime 6 + 1</td>
</tr>
</tbody>
</table>

Do not use SOS-constraints

Slack variables can be written as a sum of binary variables. If this is not done, this does not have much effect on solving the first conflict, but for the second it makes a large difference.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>151</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>136</td>
</tr>
<tr>
<td>2</td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 287/456 (62.9%)
Number of adapted constraints: 6
Number of dependent constraints changed: 24

The constraints that are changed are shown below.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H2</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>1</td>
<td>H2</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>1</td>
<td>T1</td>
<td>Odz</td>
<td>Hglo</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>1</td>
<td>T1</td>
<td>Go</td>
<td>Mk</td>
<td>Triptime 4 + 1</td>
</tr>
<tr>
<td>1</td>
<td>T1</td>
<td>Mk</td>
<td>Lc</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>2</td>
<td>815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Mp</td>
<td>Triptime 6 + 1</td>
</tr>
</tbody>
</table>

Do not provide cuts on the $p$-values

Providing extra cuts to the MIP model appeared to have no effect. If they are left out, solving the second conflict could be done a lot faster.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>150</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 173/231 (74.9%)
Number of adapted constraints: 6
Number of dependent constraints changed: 25

The constraints that are changed are shown below.
APPENDIX C. DETAILED RESULTS

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Go</td>
<td>Ddn</td>
<td>Triptime 9 + 3</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 6 - 1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>Ddn</td>
<td>Go</td>
<td>Triptime 9 + 2</td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Asnz</td>
<td>Bl Triptime 8 + 1</td>
</tr>
</tbody>
</table>

Give equal branching priorities to the \( p \)-values

The results shown below show that giving the \( p \) variables equal branching priority reduce the computation times very much.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 67/218 (30.7%)
Number of adapted constraints: 7
Number of dependent constraints changed: 31

The constraints that are changed are shown below.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Go</td>
<td>Ddn</td>
<td>Triptime 9 + 3</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 6 - 1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>Odz</td>
<td>Hglo</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>Hglgp</td>
<td>Ddn</td>
<td>Triptime 5 + 1</td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Mp</td>
<td>Ddv Triptime 6 + 1</td>
</tr>
</tbody>
</table>

Give no branching priorities

Skipping all branching priorities appear to be a lot better than providing them.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td></td>
<td>87</td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 81/231 (35.1%)
Number of adapted constraints: 6
Number of dependent constraints changed: 22

The constraints that are changed are shown below.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Go</td>
<td>Ddn</td>
<td>Triptime 9 + 3</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 6 - 1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>Ddn</td>
<td>Go</td>
<td>Triptime 9 + 2</td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Mp</td>
<td>Ddv Triptime 6 + 1</td>
</tr>
</tbody>
</table>
C.1. RESULTS ON FINDING BEST CONFLICT GRAPH

Require departure times to be integer

Leaving out the requirement to have integer departure times provides a very small effect. Probably CPLEX branches on those variables sometimes, while they will be integer in the final solution.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>746</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 237/280 (84.6%)
Number of adapted constraints: 9
Number of dependent constraints changed: 37

The constraints that are changed are shown below.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 6 + 2</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 6 + 2</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 6 + 1</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>Odz</td>
<td>Hglo</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>Hglo</td>
<td>Hgl</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>Mk</td>
<td>Lc</td>
<td>Triptime 6 + 3</td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Mp</td>
<td>Triptime 6 + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ddv</td>
<td></td>
</tr>
</tbody>
</table>

Provide a little perturbation to the objective coefficients

Symmetry breaking by using perturbations was useless. It has the effect that solutions that are equally good, are no longer seen as being equally good. During the solving process this might help, but proving optimality becomes hard as well.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>26</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>100</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>967</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>746</td>
<td></td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 218/262 (83.2%)
Number of adapted constraints: 8
Number of dependent constraints changed: 40

The constraints that are changed are shown below.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Lc</td>
<td>Mk</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Mk</td>
<td>Go</td>
<td>Triptime 7 + 2</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>Go</td>
<td>Ddn</td>
<td>Triptime 9 + 3</td>
</tr>
<tr>
<td></td>
<td>H2</td>
<td>Lr</td>
<td>Lc</td>
<td>Triptime 6 - 1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>Odz</td>
<td>Hglo</td>
<td>Triptime 7 + 1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>Hglo</td>
<td>Hgl</td>
<td>Triptime 8 + 1</td>
</tr>
<tr>
<td>2</td>
<td>260</td>
<td>T2</td>
<td>Hgv</td>
<td>Triptime 6 + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Kg</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mp</td>
<td>Triptime 6 + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ddv</td>
<td></td>
</tr>
</tbody>
</table>
C.1.2 Result on NL13new

Here the results are shown for the NL13new instance. For each conflict, the first row shows the number of nodes and arcs that CADANS/Conflex returns. This is the minimal conflict. The next line shows the size of the conflict of all dependent arcs are added. The third line shows the size of $G_L$. The last line shows the size of the conflict that is used for the final solution of the conflict.

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>32</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>227</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td></td>
<td>547</td>
<td>3754</td>
<td></td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>57</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>650</td>
<td></td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>650</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>48</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td></td>
<td>205</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>205</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>67</td>
<td>68</td>
<td>760</td>
</tr>
<tr>
<td></td>
<td>206</td>
<td>617</td>
<td></td>
</tr>
<tr>
<td></td>
<td>489</td>
<td>3592</td>
<td></td>
</tr>
<tr>
<td></td>
<td>206</td>
<td>685</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>17</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>285</td>
<td>1228</td>
<td></td>
</tr>
<tr>
<td></td>
<td>285</td>
<td>1228</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>43</td>
<td>44</td>
<td>805</td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>428</td>
<td></td>
</tr>
<tr>
<td></td>
<td>463</td>
<td>3587</td>
<td></td>
</tr>
<tr>
<td></td>
<td>199</td>
<td>815</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>43</td>
<td>44</td>
<td>693</td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td></td>
<td>373</td>
<td>2922</td>
<td></td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>496</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>173</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>29</td>
<td>30</td>
<td>1054</td>
</tr>
<tr>
<td></td>
<td>139</td>
<td>297</td>
<td></td>
</tr>
<tr>
<td></td>
<td>365</td>
<td>2878</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>603</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>57</td>
<td>74</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>84</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>482</td>
<td></td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>482</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>64</td>
<td>79</td>
<td>782</td>
</tr>
<tr>
<td></td>
<td>101</td>
<td>403</td>
<td></td>
</tr>
<tr>
<td></td>
<td>181</td>
<td>1130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>127</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>90</td>
<td>105</td>
<td>704</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td></td>
<td>164</td>
<td>1022</td>
<td></td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>702</td>
<td></td>
</tr>
</tbody>
</table>
C.1. RESULTS ON FINDING BEST CONFLICT GRAPH

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Asra</td>
<td>Shl</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
<td>Trip 13+2</td>
</tr>
<tr>
<td>11</td>
<td>H1</td>
<td>Rlbaz</td>
<td>Hszha</td>
<td>Trip 7+2</td>
</tr>
<tr>
<td>11</td>
<td>H1</td>
<td>Hsbdg</td>
<td>Brecht</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>11</td>
<td>H1</td>
<td>Brecht</td>
<td>Atwg</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>2</td>
<td>H1</td>
<td>Bhx</td>
<td>Gr</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>745</td>
<td>H1</td>
<td>Mkbr</td>
<td>Ldmw</td>
<td>Trip 4+1</td>
</tr>
<tr>
<td>745</td>
<td>T1</td>
<td>Gr</td>
<td>Bhx</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>745</td>
<td>H2</td>
<td>Bhx</td>
<td>Gr</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>745</td>
<td>H2</td>
<td>Mkbr</td>
<td>Ldmw</td>
<td>Trip 4+1</td>
</tr>
<tr>
<td>745</td>
<td>T2</td>
<td>Gr</td>
<td>Bhx</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>3</td>
<td>H1</td>
<td>Tbg</td>
<td>Vsv</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>802</td>
<td>H1</td>
<td>Vsv</td>
<td>Atn</td>
<td>Trip 7+2</td>
</tr>
<tr>
<td>802</td>
<td>T1</td>
<td>Vsv</td>
<td>Tbg</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>802</td>
<td>T1</td>
<td>Gdr</td>
<td>Dtc</td>
<td>Trip 5+2</td>
</tr>
<tr>
<td>802</td>
<td>T1</td>
<td>Dtch</td>
<td>Wl</td>
<td>Trip 4+1</td>
</tr>
<tr>
<td>802</td>
<td>T2</td>
<td>Vsv</td>
<td>Tbg</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>802</td>
<td>T2</td>
<td>Gdr</td>
<td>Dtc</td>
<td>Trip 5+2</td>
</tr>
<tr>
<td>802</td>
<td>H2</td>
<td>Tbg</td>
<td>Vsv</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>802</td>
<td>H2</td>
<td>Vsv</td>
<td>Atn</td>
<td>Trip 7+2</td>
</tr>
<tr>
<td>802</td>
<td>T2</td>
<td>Dtch</td>
<td>Wl</td>
<td>Trip 4+1</td>
</tr>
<tr>
<td>4</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
<td>Trip 13+1</td>
</tr>
<tr>
<td>020</td>
<td>T1</td>
<td>Atwg</td>
<td>Brecht</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>020</td>
<td>T1</td>
<td>Wspl</td>
<td>Hfdm</td>
<td>Trip 11+2</td>
</tr>
<tr>
<td>020</td>
<td>T1</td>
<td>Atwg</td>
<td>Brecht</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>020</td>
<td>T1</td>
<td>Hszha</td>
<td>Rlbaz</td>
<td>Trip 6+2</td>
</tr>
</tbody>
</table>

Time spend on CPLEX/Total time (sec): 9847/10046 (98.0%)
Number of adapted constraints: 71
Number of dependent constraints changed: 436

The table below shows the constraints that are changed and the way in which they are changed.
APPENDIX C. DETAILED RESULTS

C.2 Results on cut generation algorithm

Since the results on *Hdr30* and *NL13* are not really different from the results seen for the algorithm that finds the best graph to use, they are not displayed. For the *NL13new* instance, the results are shown where the algorithm is used that used $G_{0,3}$ as the largest graph.

In the first table, for each conflict, the first row shows the number of nodes and arcs that CADANS/Conflex returns. This is the minimal conflict. The next line shows the size of the conflict of all dependent arcs are added. The third line shows the size of $G_{0,3}$. The last line shows the size of the conflict that is used for the final solution of the conflict.
### C.2. RESULTS ON CUT GENERATION ALGORITHM

<table>
<thead>
<tr>
<th>Conflict</th>
<th># nodes</th>
<th># arcs</th>
<th># seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>227</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>57</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>438</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>315</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>54</td>
<td>59</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>54</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>47</td>
<td>51</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>314</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>48</td>
<td>52</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>315</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>56</td>
<td>61</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>345</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>41</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>67</td>
<td>68</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>206</td>
<td>617</td>
<td></td>
</tr>
<tr>
<td></td>
<td>229</td>
<td>728</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>36</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>268</td>
<td></td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>19</td>
<td>67</td>
<td>68</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>229</td>
<td>728</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>36</td>
<td>36</td>
<td>233</td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>16</td>
<td>17</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>92</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>17</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>92</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>43</td>
<td>44</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>206</td>
<td>550</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>42</td>
<td>43</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>170</td>
<td>525</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>91</td>
<td>111</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>696</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>43</td>
<td>44</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>169</td>
<td>555</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>57</td>
<td>74</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>98</td>
<td>369</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>51</td>
<td>68</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>74</td>
<td>96</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>432</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>64</td>
<td>79</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>127</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>40</td>
<td>41</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>76</td>
<td>404</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>79</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>127</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>90</td>
<td>105</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>826</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>76</td>
<td>91</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>117</td>
<td>648</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>76</td>
<td>91</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>694</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>68</td>
<td>83</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>117</td>
<td>648</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>197</td>
<td>212</td>
<td>431</td>
</tr>
<tr>
<td></td>
<td>432</td>
<td>1500</td>
<td></td>
</tr>
</tbody>
</table>
C.2. Results on Cut Generation Algorithm

| Time spend on CPLEX/Total time (sec): | 1068/1452 (73.6%) |
| Number of adapted constraints: | 109 |
| Number of dependent constraints changed: | 425 |

The table below shows the constraints that are changed and the way in which they are changed.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Train</th>
<th>From</th>
<th>To</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H1</td>
<td>Asra</td>
<td>Shl</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>1</td>
<td>H1</td>
<td>Hdlm</td>
<td>Wspl</td>
<td>Trip 13+2</td>
</tr>
<tr>
<td>1</td>
<td>H1</td>
<td>Lbaz</td>
<td>Hszha</td>
<td>Trip 7+2</td>
</tr>
<tr>
<td>1</td>
<td>H1</td>
<td>Hsbdg</td>
<td>Brecht</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>1</td>
<td>H1</td>
<td>Brecht</td>
<td>Atwg</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>2</td>
<td>H1</td>
<td>Blhx</td>
<td>Gr</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>2</td>
<td>H2</td>
<td>Blhx</td>
<td>Gr</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>3</td>
<td>H1</td>
<td>Tbg</td>
<td>Vsv</td>
<td>Trip 6+2</td>
</tr>
<tr>
<td>3</td>
<td>H2</td>
<td>Ahgo</td>
<td>Wtv</td>
<td>Trip 3+1</td>
</tr>
<tr>
<td>3</td>
<td>H2</td>
<td>Wtv</td>
<td>Dvn</td>
<td>Trip 3+1</td>
</tr>
<tr>
<td>3</td>
<td>H2</td>
<td>Dvn</td>
<td>Zv</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>3</td>
<td>H2</td>
<td>Zv</td>
<td>Did</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>Atn</td>
<td>Vsv</td>
<td>Trip 7+2</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>Vsv</td>
<td>Tbg</td>
<td>Trip 6+2</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>Gdr</td>
<td>Dtc</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>4</td>
<td>T1</td>
<td>Atn</td>
<td>Vsv</td>
<td>Trip 7+1</td>
</tr>
<tr>
<td>5</td>
<td>H1</td>
<td>Vsv</td>
<td>Atn</td>
<td>Trip 7+1</td>
</tr>
<tr>
<td>5</td>
<td>T1</td>
<td>Ww</td>
<td>Atn</td>
<td>Trip 9+1</td>
</tr>
<tr>
<td>6</td>
<td>T2</td>
<td>Wv</td>
<td>Atn</td>
<td>Trip 9+1</td>
</tr>
<tr>
<td>7</td>
<td>T2</td>
<td>Atn</td>
<td>Vsv</td>
<td>Trip 7-1</td>
</tr>
<tr>
<td>8</td>
<td>T2</td>
<td>Vsv</td>
<td>Tbg</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>9</td>
<td>T2</td>
<td>Ww</td>
<td>Atn</td>
<td>Trip 9+0</td>
</tr>
<tr>
<td>10</td>
<td>T2</td>
<td>Gdr</td>
<td>Dtc</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>11</td>
<td>T2</td>
<td>Gdr</td>
<td>Dtc</td>
<td>Trip 5+0</td>
</tr>
<tr>
<td>12</td>
<td>H1</td>
<td>Did</td>
<td>WI</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>12</td>
<td>T1</td>
<td>Vsv</td>
<td>Tbg</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>13</td>
<td>H2</td>
<td>Did</td>
<td>WI</td>
<td>Trip 7-1</td>
</tr>
<tr>
<td>13</td>
<td>T1</td>
<td>Dtc</td>
<td>WI</td>
<td>Trip 4+1</td>
</tr>
<tr>
<td>14</td>
<td>T2</td>
<td>Wl</td>
<td>Did</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>15</td>
<td>T2</td>
<td>Wl</td>
<td>Did</td>
<td>Trip 6+0</td>
</tr>
<tr>
<td>16</td>
<td>H2</td>
<td>Dvn</td>
<td>Zv</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>18</td>
<td>11</td>
<td>H1</td>
<td>Asdm</td>
<td>Asd</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>21</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>22</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>23</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>24</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>25</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>26</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>27</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>28</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>29</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>30</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>31</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>32</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td>33</td>
<td>11</td>
<td>H1</td>
<td>Hfdm</td>
<td>Wspl</td>
</tr>
<tr>
<td></td>
<td>H1</td>
<td>H2</td>
<td>Vama Bl</td>
<td>Bl Asnz</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>----</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>34</td>
<td>260</td>
<td>H1</td>
<td>Trip 4+1</td>
<td>Trip 6+3</td>
</tr>
<tr>
<td>35</td>
<td>260</td>
<td>H1</td>
<td>Trip 4+1</td>
<td>Trip 6+3</td>
</tr>
<tr>
<td>36</td>
<td>260</td>
<td>H2</td>
<td>Trip 4+1</td>
<td>Trip 6+3</td>
</tr>
<tr>
<td>37</td>
<td>260</td>
<td>H2</td>
<td>Trip 5+1</td>
<td>Trip 5+2</td>
</tr>
<tr>
<td>38</td>
<td>50</td>
<td>T1</td>
<td>Em Zvtwa Trip 6+1</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>210</td>
<td>H1</td>
<td>Rsd Etn Trip 8+1</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>40</td>
<td>849</td>
<td>H1</td>
<td>Zww Bp Trip 6+2</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td></td>
<td>851</td>
<td>H2</td>
<td>Bp Gk Trip 6+1</td>
<td>Trip 6+1</td>
</tr>
<tr>
<td>41</td>
<td>849</td>
<td>T1</td>
<td>Gk Bp Trip 7+1</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td>42</td>
<td>849</td>
<td>T2</td>
<td>Bp Zww Trip 5+1</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>745</td>
<td>H1</td>
<td>Bp Gk Trip 6+2</td>
<td>Trip 4+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H2</td>
<td>Vwd Zww Trip 4+1</td>
<td>Trip 5+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T1</td>
<td>Gr Bhx Trip 5+1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T2</td>
<td>Gr Bhx Trip 5+1</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Practical example Amf - Zl - Dv

In this chapter, a timetable will be developed for a largely simplified network for which a feasible solution exists. All the constraints will be derived that are relevant. In order to stick to the methodology in this thesis, trip times will be assumed to be fixed and a model concerning only departure times will be considered. Finally, a timetable will be given for this network.

D.1 Problem description

The practical example that is to be modelled is described with all the constraints that have to be satisfied.

D.1.1 Overview of the network

The network that is consider, is a part of the Dutch network. It considers the lines Amersfoort-Zwolle, Amersfoort-Deventer and Deventer-Zwolle. This network is shown in Figure D.1. The part of the network between Olst and Deventer has only single track, so trains in both directions use the same track. Further, train paths around Deventer cross each other.

![Figure D.1: Overview of the network](image)

On this network, several trains are driven, that should make certain stops and connections. The full line plan is shown in Table D.1. The first column denotes the type of the train. The next column shows a line number. The third column displays the route that it takes, with the intermediate

<table>
<thead>
<tr>
<th>Type</th>
<th>Line</th>
<th>Route</th>
</tr>
</thead>
</table>

...
stops that are relevant for this line shown in column four. In the fifth column the trip times are shown. For the 100 series, it means that it needs 18 minutes to drive Amf-Hwk, and 17 minutes to drive Hwk-Zl. In the last column, the frequency is shown. This means the number of times this train drives per hour. The regional trains will stop more often than is mentioned here, but to keep the problem small, they are not considered here.

<table>
<thead>
<tr>
<th>Type</th>
<th>Series number</th>
<th>Route</th>
<th>Relevant stops</th>
<th>Min. travel time</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercity</td>
<td>100</td>
<td>Amf-Zl</td>
<td>-</td>
<td>18+17</td>
<td>4</td>
</tr>
<tr>
<td>Intercity</td>
<td>200</td>
<td>Amf-Dv</td>
<td>Apd</td>
<td>24+10</td>
<td>2</td>
</tr>
<tr>
<td>Intercity</td>
<td>300</td>
<td>Zl-Dv</td>
<td>Ost</td>
<td>15+7</td>
<td>2</td>
</tr>
<tr>
<td>Regional</td>
<td>600</td>
<td>Amf-Zl</td>
<td>Hwk</td>
<td>24+22</td>
<td>2</td>
</tr>
<tr>
<td>Regional</td>
<td>700</td>
<td>Amf-Dv</td>
<td>Apd</td>
<td>27+12</td>
<td>2</td>
</tr>
<tr>
<td>Cargo</td>
<td>900</td>
<td>Amf-Zl</td>
<td>-</td>
<td>22+20</td>
<td>2</td>
</tr>
</tbody>
</table>

Table D.1: Line plan for the triangular network

An hourly pattern is wanted for this network. However, since all trains drive at least twice, we can change the cycle time to 30 minutes, since the next 30 minutes will be the same. This simplifies the problem quite a bit.

### D.1.2 Requirements on the timetable

**Run/dwell time** The running times mentioned in Table D.1 have to be respected. Furthermore, trains have to dwell at stations for at least 1 minute and at most 3 minutes.

**Headway time** Between each pair of trains on the network sharing a certain track, there should be at least 3 minutes.

**Deventer** The trains coming from Apd to Dv cross the trains from Dv to Ost and vice versa. Therefore, the time between any pair of an arrival and a departure should be at least 3 minutes. Furthermore, it is required that between any pair of arrivals and any pair of departure from Dv there should be at least three minutes.

**Single track** The part of the network between Dv and Ost has single track. Therefore trains can not meet each other on this part. In addition, the time between an arrival at Ost from Dv and a departure in the other direction should be at least one minute. The same holds for Dv, a train can only leave to Ost once the train coming from Ost has arrived at least one minute before.

**Transfers** In Dv, there is a possibility to transfer from the 200 series from Apd to the 300 series towards Zl and vice versa. Further, in Zl there is a possibility to transfer from the 300 series from Ost to the 600 series to Amf. In both cases, the connecting train should leave between 4 and 10 minutes after the feeder train has arrived.

**Rolling stock** Zwolle is the end station for the 300 and the 600 series. Rolling stock will serve the line in the backward direction. Turn around times are required to be at least six and at most 20 minutes.

**Absolute times** In order to fit the timetable with the rest of the network, certain times are fixed. The trains from the 100 series will depart from Amf in the intervals $[6, 10]$ and $[9, 13]$ respectively. The arrival times in Amf for the backward direction are $[15, 19]$ and $[12, 16]$ respectively.

The 600 series departs from Amf in the interval $[20, 24]$. The arrival time at Amf from the backward direction is in the interval $[0, 4]$.

The 200 series to Dv departs from Amf in $[0, 4]$ and the backward train arrives in Amf in the interval $[21, 25]$.
D.2 Derivation of the constraints

All the relevant constraints will be derived below. All will be arrival time independent and only consider departure times. Since the 100 series drive twice within the cycle period, they will be denoted by 100 and 101 respectively. The forward direction will be denoted by \( F \) and the backward by \( B \). A departure from station \( s \) of series \( t \) in direction \( h \) will be denoted by \( d_{s,t}^{h} \).

D.2.1 Trip-dwell constraints

Trip time constraints form the basis of the timetabling model. The time between two different departures is the trip time plus some dwell time. However, if a train makes the last trip of a line, for example Amf-Dv, the trip time for this last part is not put into a constraint, since there is dwell time involved at the last station. Except if there is a rolling stock connection for example.

In total the following equations are build for the trip-dwell constraints:

\[
\begin{align*}
&d_{H_{wk}}^{101,F} - d_{Amf}^{101,F} \in [18, 18]_{30} & d_{H_{wk}}^{101,B} - d_{Zl}^{101,B} \in [17, 17]_{30} \\
&d_{H_{wk}}^{102,F} - d_{Amf}^{102,F} \in [18, 18]_{30} & d_{H_{wk}}^{102,B} - d_{Zl}^{102,B} \in [17, 17]_{30} \\
&d_{Apd}^{200,F} - d_{Amf}^{200,F} \in [25, 27]_{30} & d_{Apd}^{200,B} - d_{Dv}^{200,B} \in [21, 23]_{30} \\
&d_{Ost}^{100,F} - d_{Zl}^{100,F} \in [16, 18]_{30} & d_{Ost}^{100,B} - d_{Dv}^{100,B} \in [8, 10]_{30} \\
&d_{H_{wk}}^{600,F} - d_{Zl}^{600,F} \in [25, 27]_{30} & d_{H_{wk}}^{600,B} - d_{Zl}^{600,B} \in [23, 25]_{30} \\
&d_{Apd}^{700,F} - d_{Amf}^{700,F} \in [28, 30]_{30} & d_{Apd}^{700,B} - d_{Dv}^{700,B} \in [13, 15]_{30} \\
&d_{H_{wk}}^{900,F} - d_{Amf}^{900,F} \in [23, 25]_{30} & d_{H_{wk}}^{900,B} - d_{Zl}^{900,B} \in [21, 23]_{30}
\end{align*}
\]

Note that for some arcs dwell time is set to zero, since the train does not stop there.

D.2.2 Headway constraints

For each pair of trains sharing a certain track, a headway constraint is required, separating them by at least 3 minutes. This means the difference between two departure events should be in the interval \([3, 27]_{30}\).

For the line Amf-Zl in the forward direction, we get for each \( s \in \{Amf, H_{wk}\} \) and for each pair \( t, t' \in \{101, 102, 600, 900\} \) the constraint

\[
d_{s}^{F} - d_{t}^{F} \in [3, 27]_{30}
\]

For the backward direction, we get for each \( s \in \{Zl, H_{wk}\} \) and each pair \( t, t' \in \{101, 102, 600, 900\} \) the constraint

\[
d_{s}^{B} - d_{t}^{B} \in [3, 27]_{30}
\]

On the line Amf-Dv series 200 and 700 share their tracks. Therefore we get the constraints

\[
\begin{align*}
&d_{Amf}^{600,F} - d_{Amf}^{600,F} \in [3, 27]_{30} & d_{Apd}^{600,F} - d_{Apd}^{600,F} \in [3, 27]_{30} \\
&d_{Dv}^{600,B} - d_{Dv}^{600,B} \in [3, 27]_{30} & d_{Apd}^{600,B} - d_{Apd}^{600,B} \in [3, 27]_{30}
\end{align*}
\]

The track between Zl-Dv is only used by train series 300, so no headway constraints are required. Note that the trip time of the 100 series for Amf-Zl differs at least six minutes from the trip time that the 600 series needs for the same track. Therefore, another constraint has to be added to prevent those trains overtaking each other:

\[
\begin{align*}
&d_{Amf}^{101,F} - d_{Amf}^{600,F} \in [9, 27]_{T} \\
&d_{Amf}^{102,F} - d_{Amf}^{600,F} \in [9, 27]_{T}
\end{align*}
\]
**D.2.3 Constraints around Deventer**

In Figure D.2 the infrastructure is shown around Deventer. The trains from Apd come from the South West and the trains from Ost come from the North West. Just before Dv their paths cross. Therefore they have to be separated in time.

![Figure D.2: Situation around Dv (Source: http://www.sporenplan.nl)](http://www.sporenplan.nl)

The constraints corresponding to this situation are between a departure and an arrival. They are wanted between departure events. Therefore, the constraints incorporate trip times. The only train coming from Ost is the 300 series. The trains from Apd are the 200 and the 700 series. This leads to the following constraints for the in-out situation:

\[
\begin{align*}
  d_{300}^{B} - d_{Dv}^{F} &\in [11, 40]_{30} \\
  d_{200}^{F} - d_{Apd}^{B} &\in [13, 42]_{30} \\
  d_{700}^{B} - d_{Dv}^{F} &\in [8, 37]_{30} \\
  d_{Dv}^{F} - d_{Ost}^{B} &\in [8, 37]_{30}
\end{align*}
\]  

(D.6)

For the out-in situation the following constraints are added:

\[
\begin{align*}
  d_{200}^{F} - d_{Apd}^{B} &\in [21, 50]_{30} \\
  d_{700}^{F} - d_{Apd}^{B} &\in [19, 48]_{30} \\
  d_{Ost}^{F} - d_{Dv}^{B} &\in [24, 53]_{30} \\
  d_{Ost}^{F} - d_{Dv}^{B} &\in [24, 53]_{30}
\end{align*}
\]  

(D.7)

Next to the in-out and out-in relations, we have the restriction that between any pair of arrivals at Deventer and any pair of departure, at least 3 minutes should be between them. This leads to in-in and out-out constraints. For the out-out relations, the following constraints are relevant:

\[
\begin{align*}
  d_{700}^{B} - d_{Dv}^{B} &\in [3, 27]_{30} \\
  d_{300}^{B} - d_{Dv}^{B} &\in [3, 27]_{30} \\
  d_{700}^{B} - d_{Ost}^{B} &\in [3, 27]_{30} \\
  d_{Dv}^{F} - d_{Ost}^{B} &\in [3, 27]_{30}
\end{align*}
\]  

(D.8)

The in-in relations depend on the trip times. This gives the following constraints:

\[
\begin{align*}
  d_{200}^{F} - d_{Apd}^{F} &\in [5, 29]_{30} \\
  d_{Apd}^{F} - d_{Ost}^{F} &\in [0, 24]_{30} \\
  d_{700}^{F} - d_{Apd}^{F} &\in [28, 52]_{30}
\end{align*}
\]  

(D.9)

**D.2.4 Single track**

For the single track between Ost and Dv, we had the extra restriction that there should be at least one minute between an arrival and a departure at both end stations. If we look from the Dv-side, the following constraints is to be added:

\[
\begin{align*}
  d_{Ost}^{F} - d_{Dv}^{B} &\in [8, 22]_{30}
\end{align*}
\]  

(D.10)
D.2. DERIVATION OF THE CONSTRAINTS

The same holds if we look from the Ost-side, since at both sides one minute is required and trip times are equal. So no extra constraint has to be added.

D.2.5 Transfers

First consider the connection at Dv. Passengers should be able to transfer from the 200 series to the 300 series and vice versa. This leads to the constraints

\[ d_{200,B}^{Dv} - d_{Apd}^{F} \in [14, 20]_T \]
\[ d_{200,B}^{Dv} - d_{Ost}^{F} \in [11, 17]_T \]  \hfill (D.11)

In Zl, passengers can transfer from the 300 to the 600 series and vice versa. This leads to

\[ d_{600,B}^{Zl} - d_{Ost}^{B} \in [19, 25]_T \]
\[ d_{600,F}^{Zl} - d_{Hwk}^{B} \in [28, 34]_T \]  \hfill (D.12)

D.2.6 Rolling stock constraints

In Zwolle, trains of the 300 and 600 serve the line in the backward direction. This leads to the constraints

\[ d_{300,H}^{Zl} - d_{Ost}^{B} \in [21, 35]_T \]
\[ d_{300,B}^{Zl} - d_{Hwk}^{F} \in [28, 42]_T \]  \hfill (D.13)

D.2.7 Absolute constraints

Some of the departure or arrival times are more or less fixed. First, consider the 600 series. Those trains leave Amf in [20, 24]. This can be modelled by adding a node called ABS for which we require that \( v_{ABS} = 0 \). Then the constraint becomes

\[ d_{600,F}^{Amf} - ABS \in [20, 24]_{30} \]  \hfill (D.14)

For the train coming from Hwk it gives

\[ d_{600,B}^{Hwk} - ABS \in [6, 10]_{30} \]  \hfill (D.15)

The same holds for the 200 series to and from Dv. This yields the constraints

\[ d_{200,F}^{Amf} - ABS \in [0, 4]_{30} \]
\[ d_{200,B}^{Apd} - ABS \in [27, 31]_{30} \]  \hfill (D.16)

For the 100 series, things are a bit more complicated. Here we have that we can either say that

\[ d_{101,F}^{Amf} - ABS \in [6, 10]_{30} \]  \hfill (D.17)
\[ d_{102,F}^{Amf} - ABS \in [9, 13]_{30} \]

or one could require

\[ d_{101,F}^{Amf} - ABS \in [9, 13]_{30} \]
\[ d_{102,F}^{Amf} - ABS \in [6, 10]_{30} \]  \hfill (D.18)

Preferably we do not want to state on forehand which one should go first. Therefore the constraints can be written as

\[ d_{101,F}^{Amf} - ABS \in [6, 10]_{30} \cup [9, 13]_{30} \]
\[ d_{102,F}^{Amf} - ABS \in [6, 10]_{30} \cup [9, 13]_{30} \]  \hfill (D.19)
Since the trains should run at least 3 minutes apart from each other, this will not lead to any infeasibility. The constraints can be rewritten to

\[
\begin{align*}
    d_{101,F,Amf} - \text{ABS} &\in [6, 13]_{30} \\
    d_{101,F,Amf} - \text{ABS} &\in [9, 40]_{30}  \\
    d_{102,F,Amf} - \text{ABS} &\in [6, 13]_{30} \\
    d_{102,F,Amf} - \text{ABS} &\in [9, 40]_{30}
\end{align*}
\]

(D.20)

The same holds for the incoming trains at Amf. This gives the constraints

\[
\begin{align*}
    d_{101,B,Hwk} - \text{ABS} &\in [27, 31]_{30} \cup [24, 28]_{30} \\
    d_{102,B,Hwk} - \text{ABS} &\in [27, 31]_{30} \cup [24, 28]_{30} \\
    d_{101,B,Hwk} - \text{ABS} &\in [24, 31]_{30} \\
    d_{102,B,Hwk} - \text{ABS} &\in [27, 58]_{30} \\
    d_{101,B,Hwk} - \text{ABS} &\in [24, 31]_{30} \\
    d_{102,B,Hwk} - \text{ABS} &\in [27, 58]_{30}
\end{align*}
\]

(D.21) (D.22)

### D.3 Calculating the timetable

Given all the constraints stated above, a timetable can be found. This is done using CPLEX. An example of the resulting timetable is shown in the time space diagrams in the Figure D.3, D.4 and D.5. The grey area in Figure D.4 indicates a single track area.

In these timetables, it is tried to maximize robustness. This is done by minimizing the deviations for the process times from the middle of the intervals of safety constraints.

![Figure D.3: Time space diagram for Amf-Zl](image)
D.3. CALCULATING THE TIMETABLE

Figure D.4: Time space diagram for Zl-Dv

Figure D.5: Time space diagram for Amf-Dv
Appendix E

Practical example Hdr-Sgn

In this chapter, the constraints will be described that arise in a small single track network. It clearly shows the difficulties that are involved in single track networks. For this network, a feasible timetable does not exist. Therefore, the constraints have to be relaxed. A solution to this is derived in Section 3.2.1.

E.1 Problem description

The network that is considered in this chapter concerns a part of the Dutch rail network that has single track. It consists of 4 stations, all having at least 2 platforms. Further, there is a bridge (Koegrasbrug) in the network which is seen as a train-path point as well. In fact, both sides of the bridge are seen as train path points. This is done to allow for including opening times of the bridge in the timetable. This is done by introducing a virtual ‘bridge train’, departing at one side of the bridge at the time it opens and arriving at the other side of the bridge when the bridge is free again. In total, six train path points are involved. Those are the following:

- Hdr: Den Helder
- Hdrz: Den Helder Zuid
- Kgs: Koegrasbrug (northern part)
- Kgs1: Koegrasburg (southern part)
- Ana: Anna Paulowna
- Sgn: Schagen

An overview of the network under consideration is shown in Figure E.1. First of all, a general sketch is shown in Figure E.1(a). In Figure E.1(b) the part between Den Helder and the Koegrasburg is shown in detail. In Figure E.1(c) the part between Koegrasbrug and Schagen is shown in detail. On this network, the series 468 and 469 are defined. Both series stop at Schagen, Anna Paulowna, Den Helder Zuid and Den Helder. 468 is an intercity train, driving between Den Helder and Schagen, twice per hour. 469 is a sprinter train, driving only once per hour. The trip times are shown in Table E.1.

The intercity trains have to leave at .29 and .59 from Den Helder. They have to drive in a 30/30 pattern, meaning there should be 30 minutes in between them. In fact, for one direction this is implied by the fact that they have to leave at .29 and .59.

E.2 Derivation of the constraints

There are three trains using the track in both directions. This corresponds to six one-direction trains. For each train, 5 departure times have to be fixed. This means finding 30 departure events in total.

In order to derive the constraints like DONS generates them, the network is traversed from Den Helder to Schagen when generating safety constraints.
Figure E.1: Overview of the network between Schagen and Den Helder (Source: http://www.sporenplan.nl)

<table>
<thead>
<tr>
<th>Track</th>
<th>Trip time</th>
<th>Track</th>
<th>Trip time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hdr-Hdrz</td>
<td>2</td>
<td>Hdrz-Hdr</td>
<td>3</td>
</tr>
<tr>
<td>Hdrz - Kgs</td>
<td>2</td>
<td>Kgs - Hdrz</td>
<td>2</td>
</tr>
<tr>
<td>Kgs - Kgs1</td>
<td>0</td>
<td>Kgs1 - Kgs</td>
<td>0</td>
</tr>
<tr>
<td>Kgs1 - Ana</td>
<td>3</td>
<td>Ana - Kgs1</td>
<td>3</td>
</tr>
<tr>
<td>Ana - Sgn</td>
<td>5</td>
<td>Sgn - Ana</td>
<td>5</td>
</tr>
</tbody>
</table>

Table E.1: Trip times for the two series
First of all, notation has to be fixed. Events are denoted as $d_{xDy}^z$, where $d$ means ‘departure’, $x \in \{8, 9\}$ denotes the series, $D$ denotes the direction (H for Hdr to Sgn, T for the backward direction) and $y \in \{1, 2\}$ denotes the first of the second train of this series.

First of all, the absolution constraints can be derived. The trains have to leave at .29 and .59, but we do not want to state on forehand which one is the first. Therefore, the constraints are written as:

\[
d_{8H1}^{Hdr} - \text{ABS} \in [29, 59]_T \cap [59, 89]_T \tag{E.1}
\]

\[
d_{8H2}^{Hdr} - \text{ABS} \in [29, 59]_T \cap [59, 89]_T \tag{E.2}
\]

In order to assure that not both trains leave at the same time but drive in a 30/30 pattern, the constraint

\[
d_{8H2}^{Hdr} - d_{8H1}^{Hdr} \in [30, 30]_T \tag{E.3}
\]

is added.

**E.2.1 Trip times**

The next thing to add are the trip time constraints. They ensure that the trips are executed right after each other. For the first train of the 468 series in the forward direction this leads to the constraints:

\[
d_{468H1}^{Hdrz} - d_{468H1}^{Hdr} \in [2, 2]_T \tag{E.5}
\]

\[
d_{468H1}^{Kgs} - d_{468H1}^{Hdrz} \in [2, 2]_T \tag{E.6}
\]

\[
d_{468H1}^{Kgs} - d_{468H1}^{Kgs} \in [0, 0]_T \tag{E.7}
\]

\[
d_{468H1}^{Ana} - d_{468H1}^{Kgs} \in [3, 3]_T \tag{E.8}
\]

\[
d_{468H1}^{Ana} - d_{468H1}^{Ana} \in [3, 3]_T \tag{E.9}
\]

Note that the trip time of Anna Paulowna to Schagen is not involved in those constraints. For the other directions and other series, the constraints are generated in the same way.

**E.2.2 Safety Hdr-Hdrz**

To generate the safety constraints, there were several situations to consider. There are out-out, in-in situations and any combination of them. Further, this is single track to constraints to prevent frontal collisions are added.

Three trains leave Den Helder, and they have to be separated by three minutes from each other, so $h_{oo} = 3$. The equation belonging to this situation is

\[
d_2 - d_1 \in [h_{oo}^{12}, T - h_{oo}^{21}]_T \tag{E.10}
\]

Hence, this gives the following constraints:

\[
d_{468H2}^{Hdr} - d_{468H1}^{Hdr} \in [3, 57]_T \tag{E.11}
\]

\[
d_{468H2}^{Hdr} - d_{468H1}^{Hdr} \in [3, 57]_T \tag{E.12}
\]

\[
d_{468H1}^{Hdr} - d_{468H1}^{Hdr} \in [3, 57]_T \tag{E.13}
\]

Further, trains enter Den Helder, and they have to be separated by three minutes as well, so $h_{ii} = 3$. The equation belonging to this situation was

\[
d_2 - d_1 \in [h_{ii}^{12} + r_1 - r_2, T - h_{ii}^{21} + r_1 - r_2]_T \tag{E.14}
\]

Hence, the following constraints have to be added:

\[
d_{468T2}^{Hdrz} - d_{468T1}^{Hdrz} \in [3, 57]_T \tag{E.15}
\]

\[
d_{468T1}^{Hdrz} - d_{468T1}^{Hdrz} \in [2, 56]_T \tag{E.16}
\]

\[
d_{469T1}^{Hdrz} - d_{468T2}^{Hdrz} \in [2, 56]_T \tag{E.17}
\]
Next, trains enter Den Helder while other leave, giving in-out and out-in relations. The constraint that has to be added to the model has the following form:

\[ d_2 - d_1 \in [-r_2 + \max\{p_{io}^{12}, 1\}, T - r_2 - p_{io}^{21}]T \]  
(E.18)

This constraint is stated if it is seen from the side of Den Helder. Of the same situation is seen from the side of Den Helder Zuid, it can be verified that this leads to the constraint

\[ d_2 - d_1 \in [r_1 + p_{io}^{12}, T + r_1 - \max\{p_{io}^{21}, 1\}]T, \]  
(E.19)

see also Figure 3.5. DONS puts those two constraints for the in-out and out-in relations. For the out-in situation, this leads to the following constraints:

\[ d_{468T1}^{Hdrz} - d_{468H1}^{Hdr} \in [-2, 54]T = [58, 114]T \]  
(E.20)

\[ d_{468T2}^{Hdrz} - d_{468H2}^{Hdr} \in [-2, 54]T = [58, 114]T \]  
(E.21)

\[ d_{468T1}^{Hdrz} - d_{469H1}^{Hdr} \in [-2, 54]T = [58, 114]T \]  
(E.22)

\[ d_{468T2}^{Hdrz} - d_{469H1}^{Hdr} \in [-2, 54]T = [58, 114]T \]  
(E.23)

\[ d_{469T1}^{Hdrz} - d_{469H1}^{Hdr} \in [-3, 53]T = [57, 113]T \]  
(E.25)

Note that a peak time of three minutes is for in-out relations. For out-in, the default of one minute is used.

For the in-out relation this leads to the constraints

\[ d_{468T1}^{Hdrz} - d_{468H1}^{Hdr} \in [5, 62]T \]  
(E.26)

\[ d_{468T2}^{Hdrz} - d_{468H2}^{Hdr} \in [5, 62]T \]  
(E.27)

\[ d_{468T1}^{Hdrz} - d_{469H1}^{Hdr} \in [6, 63]T \]  
(E.28)

\[ d_{468T2}^{Hdrz} - d_{469H1}^{Hdr} \in [6, 63]T \]  
(E.29)

\[ d_{469T1}^{Hdrz} - d_{469H1}^{Hdr} \in [6, 63]T \]  
(E.30)

Finally, there are constraints to avoid having frontal collisions, they have the following form:

\[ d_2 - d_1 \in [r_1 + p_{io}^{12}, T - r_2 - p_{io}^{21}]T \]  
(E.33)

For this situation, they become the following:

\[ d_{468T1}^{Hdrz} - d_{468H1}^{Hdr} \in [5, 54]T \]  
(E.34)

\[ d_{468T2}^{Hdrz} - d_{468H2}^{Hdr} \in [5, 54]T \]  
(E.35)

\[ d_{468T1}^{Hdrz} - d_{469H1}^{Hdr} \in [6, 54]T \]  
(E.36)

\[ d_{468T2}^{Hdrz} - d_{469H1}^{Hdr} \in [6, 54]T \]  
(E.37)

\[ d_{469T1}^{Hdrz} - d_{469H1}^{Hdr} \in [6, 53]T \]  
(E.38)

For the other parts of this network, the derivation goes in the same way as for this part. This would lead to a long list of constraints.
Appendix F

Map of the Dutch railway network in 2015

On the next page, a map of the Netherlands is shown. It is with permission taken from the website http://www.treinreiziger.nl. On this map, all train lines that are in operation in the timetable of 2015 are shown schematically, each line with its colour. It contains all the train lines that are in operation in the timetable 2015. If a train drives four times per hour, the line is shown is larger. If it drives only once per hour, it is shown smaller.
Bibliography


BIBLIOGRAPHY


Summary

Public transport is a good alternative for the car if it provides fast and reliable transport. One of the most important things to ensure a good public transport system, is to have a good timetable. This timetable can be used by the customer to check how often and at what point in time a train leaves. Designing a timetable is a very complex puzzle. There are many trains on the network and they are subject to many requirements, like having a short trip time and good transfer possibilities.

For many years the solver CADANS has been in use at Netherlands Railways, to assist in the development of a timetable. This algorithm solves a mathematical model that contains all requirements that a timetable should satisfy. If a timetable exists that satisfies all the constraints, CADANS finds one in a short time. However, if the set of constraints is such that no timetable exists that satisfies all the constraints, the user only gets a minimal set of constraints that can not be satisfied at the same time. Some of those constraints have to be changed in order to have a feasible solution. It is hard to understand this set of constraints and what action needs to be taken to make it feasible.

In this thesis, two algorithms are proposed that find a timetable, even if initially no feasible timetable exists. This timetable satisfies all safety constraints and stays as close as possible to the original set of constraints. Some of the constraints might be adapted in order to guarantee the existence of a feasible timetable. This algorithm repeatedly runs the CADANS solver. If this solver finds a conflict, a Mixed Integer Programming model is used to solve the conflict automatically in a good way. This MIP model finds the minimal deviations from the original set of constraints. Those constraints are adapted in such a way that the conflict is solved. Finally, CADANS is run again. In the end, unless there are too many trains on a track, a feasible timetable comes out.

The proposed algorithms provide a functional extension to CADANS. There is an option within CADANS that shows how much a certain bound of some constraint has to be relaxed to solve a conflict. The proposed algorithms make a more balanced choice. Next to that, they take into account the fact that bounds of some constraints depend on the trip time of a train on some track. If those trip times change, those constraints are changed as well. With this extension, in fact two of the shortcomings of CADANS are solved. The first shortcoming that is solved is the fact that no timetable comes out if

The first is the fact that dealing with the returned conflicts is hard. The second is that it in some sense introduced flexible trip times.

If a network contains only few conflicts, a timetable is found in only a few minutes. This is typically the case if an extra train is added to an existing timetable. For a timetable that is to be designed from scratch, it takes less than three hours to find a feasible timetable.
Samenvatting

Openbaar vervoer is een goed alternatief voor de auto, mits het snelle en betrouwbare verbindingen aanbiedt. Een van de voorwaarden voor een goed en betrouwbare openbaar vervoer systeem is een dienstregeling. Hierin kan de klant zien hoe vaak en wanneer een bepaalde trein rijdt. Het ontwerpen van een dienstregeling is een complexe puzzel. Er rijden veel treinen op het spoor en men wil goede aansluitingen tussen treinen realiseren, evenals korte rijtijden.

Al twintig jaar is het CADANS algoritme in gebruik bij de Nederlandse Spoorwegen, om te assisteren bij het ontwerpen van een dienstregeling. Dit algoritme lost een wiskundig model op met eisen waar een dienstregeling aan moet voldoen. Als er een dienstregeling bestaat die aan alle eisen voldoet, vindt CADANS deze erg snel. Als de eisen echter zodanig gesteld zijn, dat er geen dienstregeling kan bestaan (wat vaak het geval is), krijgt de gebruiker een lijst met eisen te zien waar niet tegelijkertijd aan voldaan kan worden. Sommige van deze eisen moet aangepast worden, wil er een toegelaten dienstregeling bestaan. Het is echter lastig om te zien welke actie ondernomen moet worden om dit conflict op te lossen en er voor te zorgen dat er een dienstregeling kan bestaan.

In deze scriptie worden twee algoritmen voorgesteld die een dienstregeling maken, ook als er aanvankelijk geen toegelaten dienstregeling bestaat die aan alle eisen voldoet. Deze dienstregeling kan sommige eisen schenden, maar blijft zo dicht mogelijk bij de originele set van eisen. Wat de algoritmen doen is herhaaldelijk CADANS draaien. Zodra een conflict gevonden wordt, wordt deze opgelost door midden van een Mixed Integer Programming model. Dit model vindt de kleinste mogelijke aanpassing van de set met eisen zodanig dat het conflict verholpen wordt. Met deze oplossing wordt de set van met eisen wat aangepast en CADANS wordt opnieuw gedraaid. Uiteindelijk resulteert dit in een dienstregeling die zo dicht mogelijk bij de originele set van eisen blijft. Tenzij er eenvoudigweg te veel treinen gepland worden op een bepaald traject, dat nooit zal gaan passen.

De voorgestelde algoritmen vormen een functionele uitbreiding voor het CADANS algoritme. Er is een optie binnen CADANS die weer kan geven hoe ver een grens van een bepaalde eis verruimt moet worden om een conflict te verhelpen. Deze algoritmen echter geven een meer gebalanceerde oplossing en houden eveneens rekening met het feit dat grenzen van sommige eisen afhankelijk zijn van de rijtijd op een bepaald traject, deze eisen worden dan ook aangepast. Met deze uitbreiding worden twee te kortkomingen van het CADANS algoritme opgelost. Allereerst geeft het een goede manier om de conflicten op te lossen, wat anders vrij lastig is. Daarnaast introduceert het het tot op zekere hoogte de mogelijkheid van het hebben van flexibele rijtijden. De voorgestelde algoritmen passen rijtijden op sommige trajecten aan als dat nodig mocht zijn.

Indien een netwerk weinig conflicten bevat, wordt dit binnen enkele minuten opgelost. Dit is typisch het geval als er een extra trein aan een dienstregeling wordt toegevoegd. Als een nieuwe dienstregeling wordt ontworpen die niet gebaseerd is op die van het voorgaande jaar, dan wordt deze gevonden binnen drie uur tijd.
Index

Block signalling system, 29
Branching, 59

CADANS, 9 23 14
Changing and/or-constraints, 35
Clock time, 26
Conflex, 45
Conflict, 10
Conflict graph, 52 57
Conflicting point, 26
Constraint graph, 37
Constraint programming, 48
Constraint types, 53
Constraint: absolute times, 28
Constraint: connection, 35
Constraint: frequency, 27
Constraint: headway, 29
Constraint: hindrance, 32
Constraint: in-out, 30
Constraint: overtaking, 32
Constraint: trip time, 27
Crew scheduling, 5
Cycle basis, 41
Cycle periodicity, 41
Cycles, 39

DONS, 9 23

Franz Edelman Award, 14

Integer Linear Program, 17
Intermediate arcs, 57

Line planning, 5

MIP model, 56
Mixed Integer Linear Program, 19

Neighbouring arcs, 58

Periodic event, 25
Periodic potential, 52
Periodicity constraint, 42
PESP, 25
PESP-constraint, 25
Platform Occupation Chart, 6
Polynomial time algorithm, 15

Relaxation, 53
Rolling stock scheduling, 5

Safety clique, 39

SAT, 14
Scheduled event, 25
SOS, 60
Special ordered set, 60
Spoorslag ’70, 4
Super series, 11
Symmetric timetable, 4
Symmetry, 60
Synchronisation, 12

Time space diagram, 6
Train-path point, 6 24

Uitbuigen, 12