UNIVERSITY OF OSLO
Faculty of Mathematics and Natural Sciences

Real-Time Railway Traffic Control on Single-Track Lines

Thesis for the degree of Master of Science

Thomas Nygreen

September 2011

Abstract

Punctuality is a key performance indicator for railway traffic, and high punctuality is vital if the railways are to be an attractive means of transport. In any railway network, delays will occur, and in order to maintain high punctuality, these delays must be dealt with in such a way that they are minimized and have a minimal impact on the rest of the traffic in the network. This is a complex task, as the train schedules are strongly intertwined, and capacity in the network is limited. Today, this task is done by human dispatchers, with some help from varying decision support systems.

I have formulated a mathematical model for trains running on a network of stations and tracks, and present an algorithm for solving to optimality the Railway Traffic Control of re-scheduling trains in real-time. The algorithm can be implemented using either dynamically added cover cuts, or a compact flow formulation for station capacity conflicts.

Testing the two approaches on three test cases with an increasing number of trains and stations, we see that the cover cuts implementation has more flexibility, better scaling properties, and is the best choice for real world implementations.

This thesis describes a model and an algorithm that can solve the Single-Track Railway Traffic Control problem to optimality in real-time.

Keywords: real-time traffic control, single-track, handling congestion, failures and downtime, mixed integer programming, optimal rescheduling, compact vs non-compact formulation

Acknowledgements

I would like to thank my supervisor Carlo Mannino for leading me into an exciting field of research, for the discussions we have had along the way, and for providing me with the real world data that connect my work to the real railways out there. Thank you for all your help and for being available at all times.

During the final stages of the thesis work, Carlo and I got in contact with Josef Noll at UNIK who got us in touch with the Technology Department at Jernbaneverket. I would like to thank Tone Norløff, Petter Andersen, John Eivind Presterud and Ole Martin Krogset at Jernbaneverket for taking the time to meet with Josef, Carlo and me, and I hope to see increased interest in and use of the opportunities that optimization technology brings to railway traffic control.

I would also like to thank my employer, Tarjei Huse, for being flexible during the final months of my thesis work, and for paying me to do interesting and challenging tasks.

To all my friends, and especially Anders Bonden and Ane Marte Rognskog, I would like to apologize for being such a boring friend lately, and for turning down social invitations in favour of evening writing sessions.

I also have to thank my parents, who planted in me the need to know how and why things work like they do. My father, Fritz Albregtsen, has also contributed with indispensable support in the final phase of the thesis work, pointing out weaknesses, asking the right questions and helping with wise suggestions.

But most of all I want to thank my dear Pernille, and express my gratitude for all your moral support, for tolerating my total lack of regular sleeping hours, for doing more than your share in the kitchen, and for critically listening to my ideas. Thank you for always being there when I need you!

Contents

List of Figures **5**

List of Tables **7**

List of Symbols **9**

1 Introduction **11**

2 A short introduction to railways **15**

2.1 Infrastructure 15

2.2 Fixed and moving blocks 16

2.3 European Rail Traffic Management System (ERTMS) 17

2.4 Measuring quality of service 18

3 Model **19**

3.1 Notation and graphs 19

3.2 Modelling railway networks 20

3.3 A model for station-track networks 20

3.3.1 The routes graph 21

3.4 Assumptions 23

3.5 Preventing conflicts 23

3.5.1 Case 1: Trains moving in opposite directions 24

3.5.2 Case 2: Trains moving in the same directions 27

3.6 Handling track sections 27

3.7 Moving blocks 28

4 Solving optimization problems **29**
4.1 Concepts from graph theory 29
4.1.1 Cliques 29
4.1.2 Interval graphs 30
4.1.3 Flows 30
4.2 Linear programming 32
4.2.1 Integer and mixed integer linear programming 32
4.2.2 Piecewise linear programs 32
4.3 Solution strategies 34
4.4 Separation oracles 34
4.5 The cutting plane method 34

5 Algorithm **35**
5.1 Evaluating the actual timetable 35
5.2 A MILP formulation for STC 36
5.3 Strategies for eliminating station conflicts 37
5.3.1 Cover cuts 37
5.3.2 A compact, flow based representation 37
5.4 Identifying possible conflicts 42

6 Results **43**
6.1 Small instance 44
6.2 Medium-sized instance 46
6.3 Large instance 47
6.4 Comparing the sizes of the instances 48
6.5 Impact of multi-threading 48

7 Discussion and Conclusion **51**
7.1 Comparing the two strategies for conflict-free solutions 51
7.2 General performance of the algorithm 52
7.3 Implementation challenges 52
7.4 Conclusion and future research 53

References **54**

List of Figures

1.1 Graphical timetable 14

2.1 A braking curve 16

3.1 Simple graph examples 20
3.2 Examples of routes graphs 22
3.3 Routes graphs with meeting trains 25
3.4 Routes graphs with disjunctive arcs 26

4.1 Interval graph 30
4.2 Augmenting a flow 31
4.3 Examples of convex piecewise linear function 33

5.1 The cost function used in the simulations 36
5.2 Interval graph for a station 38
5.3 The support network 38
5.4 The cut in the necessity proof. 40

6.1 Constraint matrix dimensions, cover cuts 49
6.2 Constraint matrix dimensions, cover cuts 49
6.3 Reduced dimensions, cover cuts 50
6.4 Reduced dimensions, cover cuts 50

List of Tables

6.1 Detailed results from running the small instance 45

6.2 Detailed results from running the medium-sized instance . . . 46

6.3 Detailed results from running the large instance 47

6.4 Running time comparison with different number of threads. Times in seconds. 48

List of symbols used, in alphabetic order

$A = \cup_{i\in T} A^i \cup \{(r, v_1^i) : i \in T\}$	The set of all arcs between nodes
A^i	The arcs in P^i
A_s^{ij}	The set of arcs added if i meets j in s
B	The set of all blocks
b	A block in B
$c_v(t_v)$	The cost function for node v
d_s	The capacity of station s
$E = E_r \cup E_U \cup E_W \cup E_p \cup \{(p, r)\}$	The arcs in $N(s, \boldsymbol{y})$
e	An arc in E
$E_p = \{(w, p) : w \in W\}$	Arcs in E for the release of a platform
$E_r = \{(r, u) : u \in U\}$	Arcs in E for the initial taking of a platform
$E_U = \{(u_j, w_j) : j \in T\}$	Arcs in E for a train occupying a platform
$E_W = \{(w_i, u_j) : j \in Su(i)\}$	Arcs in E for the overtaking of a platform
f_e	The upper bound of flow on e
$G^T = (V, A)$	The routes graph
$G(\boldsymbol{y}) = (V, A \cup \{\cup_{y_s^{ij}=1} A_s^{ij}\})$	The graph obtained by adding A_s^{ij} to G^T when $y_s^{ij} = 1$
H	A cut in $N(s, \boldsymbol{y})$
i	A train in T
j	A train in T
$K \subset \{\{i, j\} : i \in T, j \in T\}$	The set of possibly conflicting trains
k	An index
$l(i)$	The number of railway resources used by i
l_e	The lower bound on e

M	A large constant
m	An index
$N(s, \boldsymbol{y}) = (\{r, p\} \cup U \cup W, E)$	Support graph for station conflict discovery
o	The end (sink) node of G^T
P^i	The path of train i
p	Sink in $N(s, \boldsymbol{y})$
q	Number of stations
$Q = \{k, k+1, \ldots, k+d_s\}$	Set of $d_s + 1$ trains meeting at s
$R = S \cup B$	The set of all railway resources
r	The root node of G^T
r	The root node of $N(s, \boldsymbol{y})$
$rr(v)$	The railway resource corresponding to v
$S = \{1, \ldots, q\}$	The set of all station
$S(ij)$	The set of stations where i and j can meet
s	a station in S
$Su(j) = Su(j, s, \bar{\boldsymbol{y}})$	The set of trains which enter s after j leaves s
T	The set of all trains
$\boldsymbol{t}(\boldsymbol{y})$	The schedule computed on $G(\boldsymbol{y})$
$t_v = t_{ik}$	The minimum time for train i to reach $v = v_k^i$
$U = \{u_1, \ldots, u_{\lvert T \rvert}\}$	Node representing a train entering the station in $N(s, \boldsymbol{y})$
u	A node in V
u_i	A node in U
$V = \cup_{i \in T} V^i \cup \{r\}$	The set of all nodes associated with the train routes
v	A node in V
V^i	The nodes in P^i
v_k^i	The k-th railway resource used by i
$W = \{u_1, \ldots, u_{\lvert T \rvert}\}$	Node representing a train leaving the station in $N(s, \boldsymbol{y})$
$W_{k,k+1}^i$	The time needed for train i to move from v_k^i to v_{k+1}^i
w_i	A node in W
x_s^{ij}	1 if $j \in Su(i, s, \boldsymbol{y})$, 0 otherwise
y_s^{ij}	1 if i and j meet at s, 0 otherwise
z_s^e	Flow on arc e in $N(s, \boldsymbol{y})$

Chapter 1

Introduction

Despite the official aim for a doubling of the freight transport volume on Norwegian railways by 2020, manifested by the Norwegian National Rail Administration (Jernbaneverket) in 2007[12] and later adopted by the Norwegian Parliament in 2009[23], the actual transport volume decreased by more than 9 percent from 2008 to 2010, returning to the lowest level since 2005. At the same time the growth in the passenger volume has stagnated, leaving the passenger numbers at the same level in 2010 as in 2008[15, p. 27]. Both logistics companies and commuters rely on the trains they use to be punctual. *Punctuality* is defined as the percentage of trains reaching their final destination on time or with a limited delay. The allowed delay varies. Jernbaneverket uses six minutes as the limit for long distance and freight trains, and four minutes for local, airport and intercity trains. Their statistics show that from May 2010 through May 2011 the monthly average punctuality varied between 83 and 91 percent for passenger trains, and between 50 and 80 percent for freight trains. The goal for passenger trains is 90 percent.[16, 17]

Delays occur for a number of reasons, including infrastructure or train failures, waiting for passengers, corresponding transports or even train personnel, or external events like land slides and avalanches. Some of these involve temporary changes to the properties of the railway network, such as reduced capacity at a station or speed reductions on certain tracks. In addition, a delayed train often interferes with other trains causing further delays. In cases where the traffic volume approaches the track capacity, even small delays may spread quickly and cause congestion. On single-track lines, a single delayed train affects other trains in both directions.

Norway has long stretches of single-track railway. Out of a total of 3950 km of lines open for traffic, only 241 km have double tracks[15]. On a single track, trains have to meet (cross or catch up) at stations or other designated meeting points. Detailed graphical timetables determine where different trains are planned to meet. A four hour excerpt for the single-track railway between Eidsvoll and Dombås on the Dovre line in Norway is shown in Figure 1.1. The graphical timetables are time-distance diagrams with plotted lines for all scheduled trains as well as information about the location of and distance between stations, crossing loops and other significant features of the infrastructure. While this line uses between 86 and 100 percent of its capacity during peak hours, several other lines in Norway have a peak usage of more than 100 percent[14]. In this case, the capacity is the maximal number of trains per hour on the line before congestion causes reduced speed and therefore increased travel time.

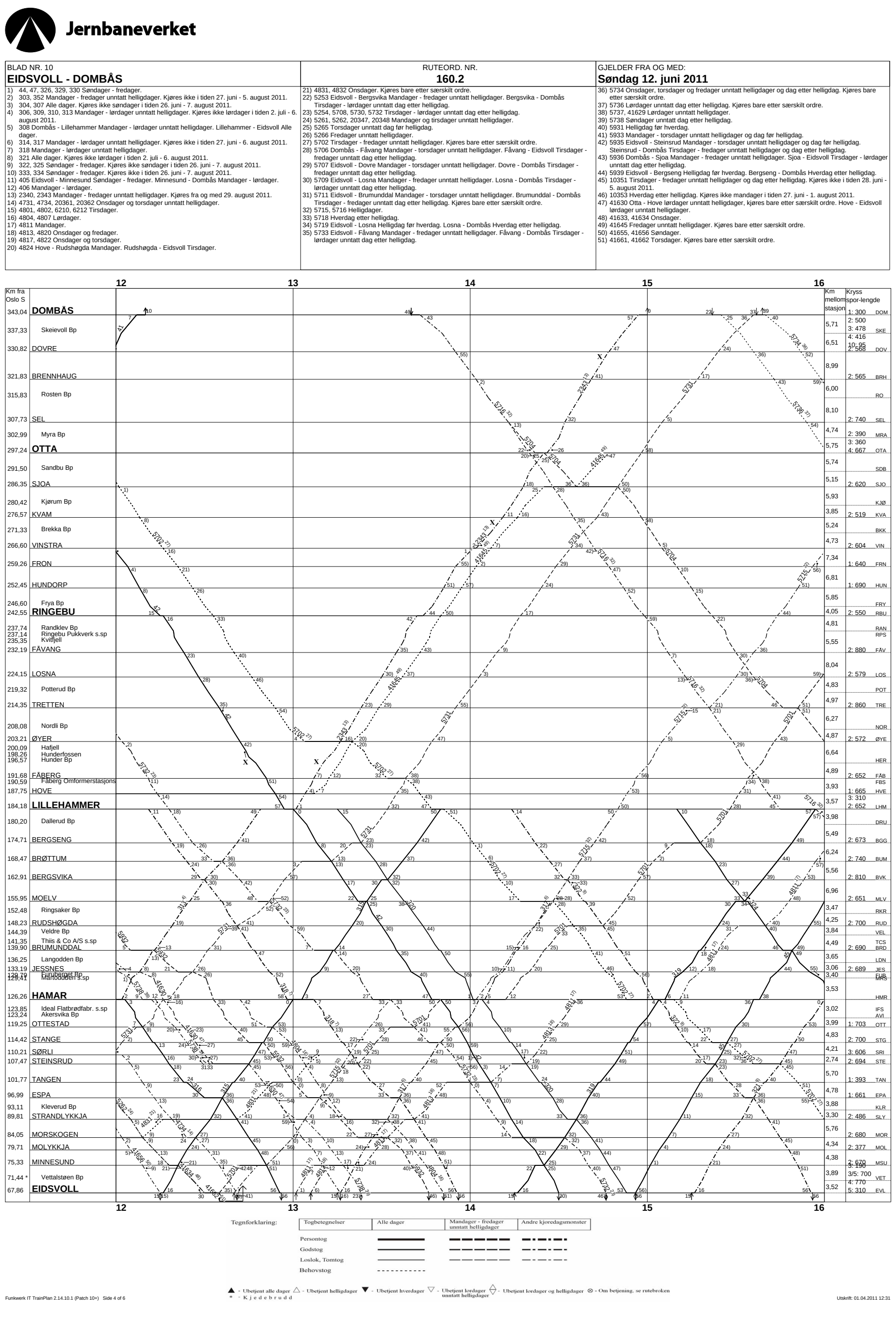
Figure 1.1 also illustrates how strongly connected the train schedules are. The pattern of where trains should meet is carefully planned each time any changes are made to the official timetables. When any delays, signal failures, power outages or other changes occur, the traffic controllers have to reconsider the planned schedule and perhaps assign pairs of trains to meet at another station, possibly delaying a train on purpose in order to let other trains reduce their delays. Delays may also be reduced by shortening the stops at stations, and in some cases there is a slack in the scheduled running time between stations, allowing trains to eliminate small delays. Ideally, the decisions made by the traffic controllers should constitute an optimal re-scheduling of the trains, minimizing (the cost of) the delays.

Traffic control of trains today is almost entirely done by human traffic controllers, called dispatchers. Implementing and deploying optimization algorithms that efficiently calculate optimal re-scheduling of trains should improve the ability of the railway networks to handle congestion and recover from unforeseen events, in practice increasing the capacity of the network.

A system for optimal real-time traffic control in metro stations was in operation on the Milan metro from 2007 to 2009. Test results showed that human dispatchers in most cases were outperformed by the automated real-time control system. Prior to deployment, the final tests at the main Sesto FS metro station showed an increase in punctuality when compared to manual control of more than 9 percentage points.[22] Several other implementations are described in [4], but to my knowledge, the only live system solving to optimality is the Lötschberg Base Tunnel system (operated by the Swiss BLS), developed by Systransis AG, which is a proprietary closed-source system.

This thesis aims to construct a suitable model and develop and implement an algorithm that in real-time can find the (re)scheduling of trains that minimizes the deviations from the planned timetable, using a convex cost function. Here, real-time means within a matter of seconds, as opposed to off-line optimization in timetable planning, where time is less restricted. The input is the layout of the railway network in its current state, taking into account any failures and outages, the planned timetable, the position of all trains and a cost function for deviations from the timetable. The task is then to solve the Railway Traffic Control problem, amounting to find a schedule for all trains that avoids resource conflicts between trains and minimizes the cost of any deviations from the planned schedule. A mixed integer linear programming (MILP) formulation is used to solve the problem of rescheduling trains to optimality, exploring two different strategies for eliminating resource conflicts. A compact flow-based representation of conflict-free solutions is derived and is compared to a simpler strategy of dynamically added cover cuts.

I will start this thesis by giving a short introduction to railway networks in Chapter 2 followed by followed by a presentation of the concept of graphs and a mathematical model for trains running through a railway network of stations and tracks in Chapter 3. Then Chapter 4 gives a short introduction to the concepts of mathematical optimization used in this thesis. Chapter 5 presents the algorithm developed to solve the Railway Traffic Control problem, while the results of simulations using the algorithm are given in Chapter 6. Finally Chapter 7 discusses these results, draws the conclusions, and lists tasks for future research.



Jernbaneverket

BLAD NR. 10	RUTEORD. NR.	GJELDER FRA OG MED:
EIDSVOLL - DOMBÅS	**160.2**	**Søndag 12. juni 2011**

1) 44, 47, 326, 329, 330 Søndager - fredager.
2) 303, 352 Mandager - fredager unntatt helligdager. Kjøres ikke i tiden 27. juni - 5. august 2011.
3) 304, 307 Alle dager. Kjøres ikke søndager i tiden 26. juni - 7. august 2011.
4) 306, 309, 310, 313 Mandager - lørdager unntatt helligdager. Kjøres ikke lørdager i tiden 2. juli - 6. august 2011.
5) 308 Dombås - Lillehammer Mandager - lørdager unntatt helligdager. Lillehammer - Eidsvoll Alle dager.
6) 314, 317 Mandager - lørdager unntatt helligdager. Kjøres ikke i tiden 27. juni - 6. august 2011.
7) 318 Mandager - lørdager unntatt helligdager.
8) 321 Alle dager. Kjøres ikke lørdager i tiden 2. juli - 6. august 2011.
9) 322, 325 Søndager - fredager. Kjøres ikke søndager i tiden 26. juni - 7. august 2011.
10) 333, 334 Søndager - fredager. Kjøres ikke i tiden 26. juni - 7. august 2011.
11) 405 Eidsvoll - Minnesund Søndager - fredager. Minnesund - Dombås Mandager - lørdager.
12) 406 Mandager - lørdager.
13) 2340, 2343 Mandager - fredager unntatt helligdager. Kjøres fra og med 29. august 2011.
14) 4731, 4734, 20361, 20362 Onsdager og torsdager unntatt helligdager.
15) 4801, 4802, 6210, 6212 Tirsdager.
16) 4804, 4807 Lørdager.
17) 4811 Mandager.
18) 4813, 4820 Onsdager og fredager.
19) 4817, 4822 Onsdager og torsdager.
20) 4824 Hove - Rudshøgda Mandager. Rudshøgda - Eidsvoll Tirsdager.
21) 4831, 4832 Onsdager. Kjøres bare etter særskilt ordre.
22) 5253 Eidsvoll - Bergsvika Mandager - fredager unntatt helligdager. Bergsvika - Dombås Tirsdager - lørdager unntatt dag etter helligdag.
23) 5254, 5708, 5730, 5732 Tirsdager - lørdager unntatt dag etter helligdag.
24) 5261, 5262, 20347, 20348 Mandager og tirsdager unntatt helligdager.
25) 5265 Torsdager unntatt dag før helligdag.
26) 5266 Fredager unntatt helligdager.
27) 5702 Tirsdager - fredager unntatt helligdager. Kjøres bare etter særskilt ordre.
28) 5706 Dombås - Fåvang Mandager - torsdager unntatt helligdager. Fåvang - Eidsvoll Tirsdager - fredager unntatt dag etter helligdag.
29) 5707 Eidsvoll - Dovre Mandager - torsdager unntatt helligdager. Dovre - Dombås Tirsdager - fredager unntatt dag etter helligdag.
30) 5709 Eidsvoll - Losna Mandager - fredager unntatt helligdager. Losna - Dombås Tirsdager - lørdager unntatt dag etter helligdag.
31) 5711 Eidsvoll - Brumunddal Mandager - torsdager unntatt helligdager. Brumunddal - Dombås Tirsdager - fredager unntatt dag etter helligdag. Kjøres bare etter særskilt ordre.
32) 5715, 5716 Helligdager.
33) 5718 Hverdag etter helligdag.
34) 5719 Eidsvoll - Losna Helligdag før hverdag. Losna - Dombås Hverdag etter helligdag.
35) 5733 Eidsvoll - Fåvang Mandager - fredager unntatt helligdager. Fåvang - Dombås Tirsdager - lørdager unntatt dag etter helligdag.
36) 5734 Onsdager, torsdager og fredager unntatt helligdager og dag etter helligdag. Kjøres bare etter særskilt ordre.
37) 5736 Lørdager unntatt dag etter helligdag. Kjøres bare etter særskilt ordre.
38) 5737, 41629 Lørdager unntatt helligdager.
39) 5738 Søndager unntatt dag etter helligdag.
40) 5931 Helligdag før hverdag.
41) 5933 Mandager - torsdager unntatt helligdager og dag før helligdag.
42) 5935 Eidsvoll - Steinsrud Mandager - torsdager unntatt helligdager og dag før helligdag. Steinsrud - Dombås Tirsdager - fredager unntatt helligdager og dag etter helligdag.
43) 5936 Dombås - Sjoa Mandager - fredager unntatt helligdager. Sjoa - Eidsvoll Tirsdager - lørdager unntatt dag etter helligdag.
44) 5939 Eidsvoll - Bergseng Helligdag før hverdag. Bergseng - Dombås Hverdag etter helligdag.
45) 10351 Tirsdager - fredager unntatt helligdager og dag etter helligdag. Kjøres ikke i tiden 28. juni - 5. august 2011.
46) 10353 Hverdag etter helligdag. Kjøres ikke mandager i tiden 27. juni - 1. august 2011.
47) 41630 Otta - Hove lørdager unntatt helligdager, kjøres bare etter særskilt ordre. Hove - Eidsvoll lørdager unntatt helligdager.
48) 41633, 41634 Onsdager.
49) 41645 Fredager unntatt helligdager. Kjøres bare etter særskilt ordre.
50) 41655, 41656 Søndager.
51) 41661, 41662 Torsdager. Kjøres bare etter særskilt ordre.

Figure 1.1: Graphical timetable (distance time graph) for the hours 12–16 on the single-track line between Eidsvoll and Dombås (275 km), from the Norwegian National Rail Administration (Jernbaneverket)[13]. Note that the vertical distance axis is not completely linear.

Chapter 2

A short introduction to railways

This chapter aims to present users with little or no prior knowledge of railway structure, terminology and standards with a brief introduction to the concepts used in this thesis.

2.1 Infrastructure

A *signalling block* is the segment of track between railway signals. When using moving blocks, the position and length of a block is not fixed (see Section 3.7 on page 28)

A *crossing loop* or *passing loop* is a meeting point on a single-track line that enables trains to pass each other.

A *movement authority* is a permission given to a train driver, allowing the train to use one or more signalling blocks.

The *breaking curve* for a moving train is the distance-velocity curve describing the deceleration of a train after the driver starts braking. See Figure 2.1 for an illustration. The braking curve for a given train changes with the speed of the train and the inclination of the track.

The *breaking point* for a moving train is the point at which it must start to slow down in order to stop before a certain point, e.g. to comply with the given movement authority. The breaking point depends on the speed and breaking curve of the train, as illustrated in Figure 2.1.

Headway is the distance in time between the front of two consecutive trains.

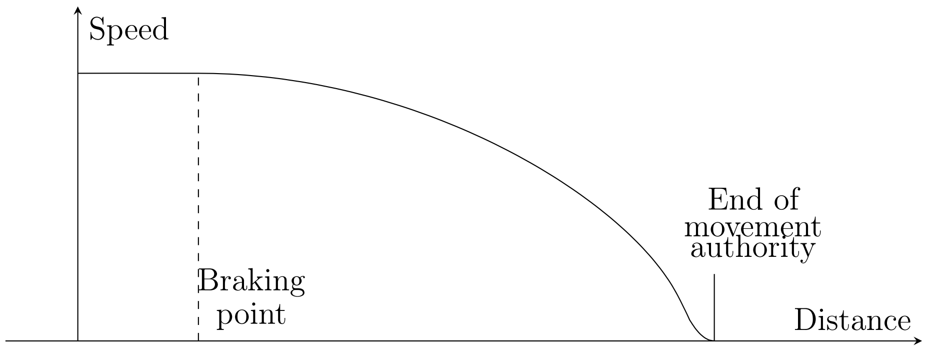


Figure 2.1: A braking curve. The slope evens out towards zero in order to avoid an uncomfortable hard stop.

Automatic Train Protection (ATP) systems apply braking power automatically if the driver fails to stop at a stop signal. Some systems also react in case the speed of the train is higher than the speed limit for the track.

Automatic Train Control (ATC) includes the features of ATP, and makes in-cab signalling to the driver possible, reducing the need for lineside signals.

Automatic train operation (ATO) allows for automatic piloting of the vehicles, as well as automatic dispatching.

2.2 Fixed and moving blocks

In traditional *fixed block* systems the boundaries between signalling blocks are fixed. These boundaries coincide with lineside signals. As each train occupies an entire block, the block length puts a lower limit on. The headway can be reduced by reducing block lengths or implementing *moving blocks*, where the blocks are calculated in real-time based on train locations.

In a *moving block* system the blocks are not fixed, but are calculated based on the exact location and speed of the trains on the track. A train is regularly given a *movement authority* allowing it to move forward up to a given point. The train itself calculates its braking curve, giving the braking distance and braking point[9, p. 44]. Basically, the moving block contains the train itself, its braking distance and security margins allowing for delays in computing and communication. Theoretically, given that consecutive trains have the same braking curve, the headway can be further reduced so that it only includes the breaking response time, but this would not include sufficient margins if the train in front crashes or derails.

2.3 European Rail Traffic Management System (ERTMS)

The European Rail Traffic Management System (ERTMS) is a project that was set up to create a common signalling and communication standard for railways throughout Europe. It consists of three parts[5, 11]:

GSM-R is a dedicated mobile radio communication standard building on the GSM mobile phone standard. It allows for uninterrupted communication at high speeds.

The European Train Control System (ETCS) is a standard for signalling, train protection and train control. It is based on a combination of on-board computers and centralized control.

The European Traffic Management Layer (ETML) is a management level intended to optimize train movements by implementing real-time re-routing and re-scheduling of trains.

ETCS specifies a four-layered standard for train control[10]:

Level 0 describes ETCS-equipped trains running on non-ETCS tracks.

Level 1 is a system connected to the existing signalling system. *Balises* along the track record information about train position, speed and integrity, and communicates movement authorities to the train as it passes.

Level 2 removes the dependency on lineside signals. Instead signals are transmitted to the trains through GSM-R. This allows trains to receive new movement authorities at any time, not only when passing a balise. Lineside track relays are still used to record the position of the trains.

Level 3 is still in a development phase. The trains themselves determine their exact position and relay it to the control central. This level also allows for moving blocks.

ETCS is adopted as a standard by the European Union[6], and is already in use on several lines, with further plans of implementing it on almost 10 000 km of lines by 2015 and around 40 000 km by 2020[7].

2.4 Measuring quality of service

Regularity is measured as the percentage of trains that run as planned (with or without delay) without being partially or fully cancelled[17].

Punctuality is measured as the percentage of trains reaching their final destination on time or with a delay of less than six minutes. For local, airport and intercity trains the delay must be less than four minutes[17].

The *uptime* is given by subtracting the number of delay hours caused by issues with the infrastructure from the total number of planned train hours for passenger and freight trains. This is then given as a percentage of the total number of planned train hours for passenger and freight trains[17].

Chapter 3

Model

This chapter details the model used to represent the railway network and trains, and the assumptions I make about the structure of this network and the planned schedules of these trains.

3.1 Notation and graphs

Bold letters (e.g. $\boldsymbol{x}$) will be used for vectors and vector valued functions, and capital letters (e.g. A) for matrices.

For n-vectors x and y the notation $\boldsymbol{x} \leq \boldsymbol{y}$ will be used to mean that $x_i \leq y_i$ for all $i = 1, \ldots, n$. Likewise $\boldsymbol{x} \geq \boldsymbol{0}$ implies that all elements in $\boldsymbol{x}$ are non-negative.

An *undirected graph* (Figure 3.1a) is a pair $G = (V, E)$ where V is a set of $|V|$ *vertices* or *nodes* and $E \subset \{\{u, v\} : u, v \in V\}$ is a set of $|E|$ *unordered* node pairs called *edges*. A pair of nodes in the graph are called *connected* if there is a continuous path between the nodes along the edges in the graph. The graph is called connected if all pairs of nodes in the graph are connected.

A *directed graph* (Figure 3.1b) is a pair $G = (V, A)$ where V is a set of $|V|$ vertices or nodes and $A \subset \{(u, v) : u, v \in V\}$ is a set of $|A|$ *ordered* node pairs called *arcs*. (u, v) is the arc from u to v. An s-t path in a directed graph is a continuous path that starts at the node s and ends at the node t obeying the directions of the arcs.

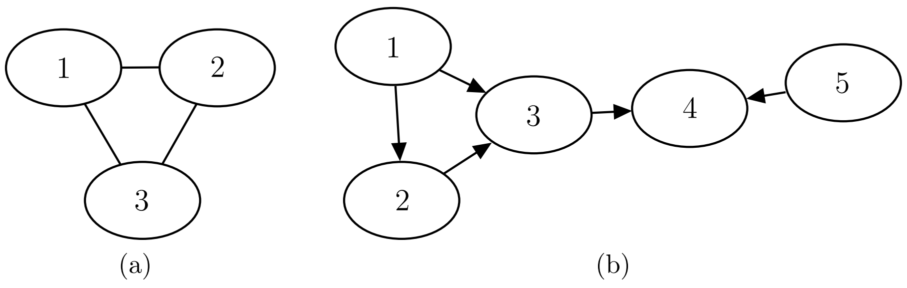


Figure 3.1: (a) An undirected graph. (b) A directed graph.

3.2 Modelling railway networks

A railway network may be modelled as a connected graph where the nodes are different types of *railway resources*, like platforms, switch-points, junctions and track segments. For safety reasons the network is normally divided into *signalling blocks*, such that no two trains can occupy the same block at the same time. Furthermore, trains running in the same direction may be restricted to having at least a certain number of empty signalling blocks between them.

This model uses two types of resources: *stations* and *blocks*. The stations may have one or more *platforms*, and are connected by blocks. As I am modelling single-track lines, each pair of two adjacent stations are connected by exactly one track. For simplicity, I do not divide the track between stations into more than one block, but the extension to multiple blocks between stations is straightforward (see Section 3.6 on page 27).

3.3 A model for station-track networks

Let $R = S \cup B$ be the set of railway resources, where S is the set of stations and B is the set of blocks. While only one train can occupy a block resource, each station $s \in S$ can accommodate up to d_s trains, where d_s is the *station capacity*. I will assume that the station layout is such that the order in which the trains may leave the station is independent of the order in which they entered.

Let T be the set of trains, and let $l(i)$ for any train $i \in T$ be the number of railway resources used by i. The route of i may be represented by a path

$P^i = \{v^i_1, (v^i_1, v^i_2), \ldots, (v^i_{l(i)-1}, v^i_{l(i)}), v^i_{l(i)}\}$ where node $v^i_k \in R$ for $1 \leq k \leq l(i)$ represents i using the k-th railway resource on its path. Since a train may already be anywhere on the line, the sequence may very well start from a block or from a specific point on the block.

Let V^i denote the set of nodes of P^i. For any $i \in T$ and $v \in V^i$, let $rr(v) \in \mathbb{R}$ denote the corresponding railway resource, and $tr(v) \in T$ the train. Furthermore, for $u \in V^i$ and $v \in V^j$ let $u \approx v$ denote that u and v correspond to the same railway resource (i.e., $rr(u) = rr(v)$). Note that $V^i \cap V^j = \emptyset$ for $i \neq j$, as the nodes in V^i and V^j for $i \neq j$ represent locations of two different trains. In order to compare paths, define $V^i|V^j$ for i and j in T to be the nodes of P^i that correspond to railway resources also used by j. Formally $V^i|V^j = \{v \in V^i : \exists u \in V^j (u \approx v)\}$.

The arcs A^i of P^i represent precedence constraints, i.e., the fact that the resource corresponding to node v^i_k is visited by the train before the resource corresponding to node v^i_{k+1}. With each arc $(u, v) = (v^i_k, v^i_{k+1}) \in A^i$ associate the weight $W_{uv} = W^i_{k,k+1} \geq 0$ representing the minimum time in seconds necessary for train i to move from the k-th resource to the next. Thus, if v^i_k represents i stopping at a station, then $W^i_{k,k+1}$ is the time the train should spend in the station before departing. If v^i_k corresponds to a block, then $W^i_{k,k+1}$ is the time needed to reach the next station (or block, if there are multiple blocks between stations).

3.3.1 The routes graph

The core of the model is the *routes graph*, as defined in this section, and shown in Figure 3.2. The routes graph $G^T = (V, A)$ is constructed by letting $A = \{(r, v^i_1), i \in T\} \cup \{(u, v) \in A^i : i \in T\}$ and $V = \{r\} \cup \{v \in V^i : i \in T\}$, that is V contains all nodes associated with the train routes plus an additional root node r and an end node o. So the new node r is a source, connected to the first node of each train route P^i, and o is a sink, connected to the end of each route.

For each $i \in T$, associate with the arc $(r, v^i_1)\}$ the weight W_{ri}, which represents the number of seconds (from now) until the train is expected to start its route. If the train is already occupying a resource $W_{ri} = 0$, as shown in Figure 3.2a. Similarly, the weight W_{oi} on $(v^i_{l(i)})$ represents the number of seconds from the train reaches its final node until it is removed from the model. This happens whenever the train leaves the physical network represented in the model (e.g. crosses over into a network not governed by the model, or is stowed away on a sidetrack).

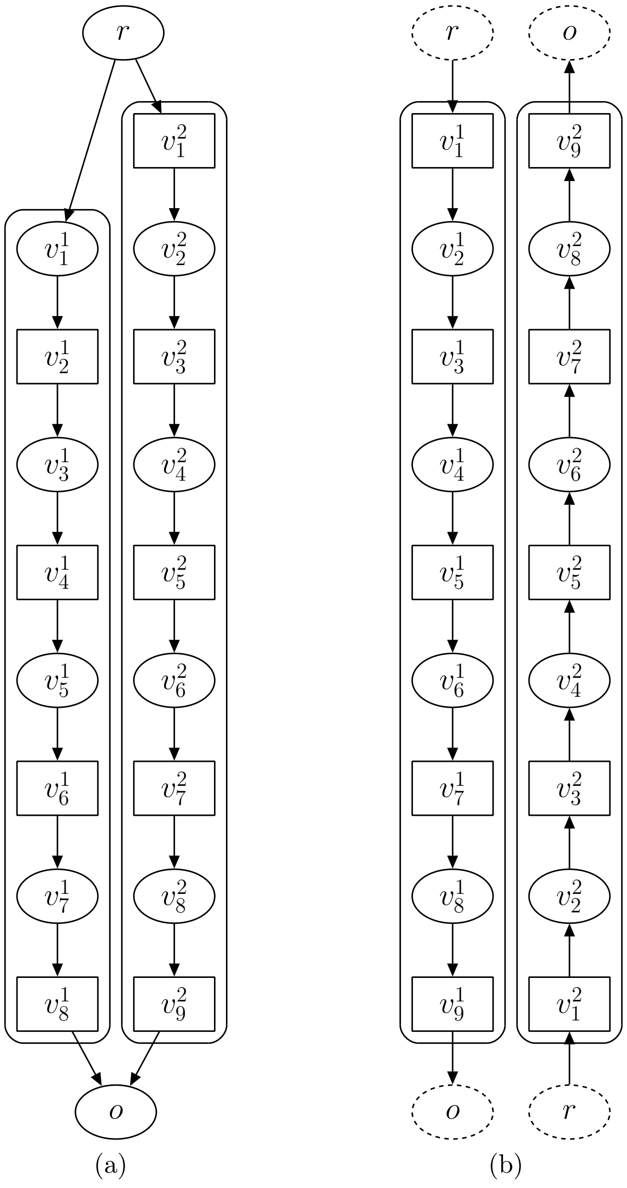


Figure 3.2: Example of routes graphs for five stations and two trains. Nodes corresponding to stations are rectangular, while nodes corresponding to blocks are oval. Nodes corresponding to the same railway resources are placed side by side. (a) shows two trains running in the same direction. This graph also displays weights on the arcs showing the minimum amount of time required to move from one resource to the next. (b) shows two trains running in opposite directions. Here, the source r and sink o have been duplicated in order to allow showing corresponding nodes side by side.

Finally, for each node $v = v^i_k$ in $V - \{r\}$ let us denote by $t_v = t_{ik}$ the minimum time in which train i can reach the k-th resource on its path. Also, let $t_r = 0$. Observe that by definition each arc $(u, v) \in A$ with weight W_{uv} represents the time constraint $t_v \geq t_u + W_{uv}$. For all $v \in V$, the quantity t_v can thus be computed by a longest-path tree computation on G^T with weights W and root r. The vector $\boldsymbol{t} \in R^V_+$ is called a *schedule* or an *actual timetable*. Indeed, if $v = v^i_k$ is a station, then t_v represents the minimum *arrival time* for train i at the station. Similarly, if $v = v^i_k$ is a block b, then t_v represents the minimum time for train i to enter the block or, equivalently, the *departing time* from the station which precedes block b on P^i. This interpretation must be slightly modified when train i is already in the line and $v = v^i_1$, that is the node is the first on P^i. In this case, we always have $t_{i1} = 0$ (since $t_r = 0$ and $W_{ri} = 0$), and t_{i2} represents the time needed for i to complete the remainder of the block.

3.4 Assumptions

In order to avoid unnecessary complexity, I make the following two assumptions. They can both be dropped, at the cost of adding new variables to the model.

1. For any two given resources, all trains using both resources follow the same path between them, possibly in reverse order. I.e., for any two trains i and j in T, the nodes $V^i|V^j$ form a continuous subpath in P^i.

2. If a train $i \in T$ overtakes another train $j \in T$, then j cannot overtake i at another station. Note that any overtakings that have already happened are invisible to the model, so i overtaking j in the past does not exclude j overtaking i, e.g. in case of trouble with train i.

3. A following train does not catch up to the train in front at any station unless it overtakes it at that station.

3.5 Preventing conflicts

The schedule $\boldsymbol{t}$ approximates the behaviour of the trains along the line. However, we need to take into account other precedence constraints in order to correctly predict the actual train timetable. It is reasonable to assume that the official timetable does not include resource conflicts between trains, but due to train delays or line dysfunctions, such conflicts may arise. There are two types of conflicts:

1. *Block conflict*: Two trains are predicted simultaneously on the same block.
2. *Station conflict*: The number of trains predicted to simultaneously be accommodated in a given station exceeds the station capacity.

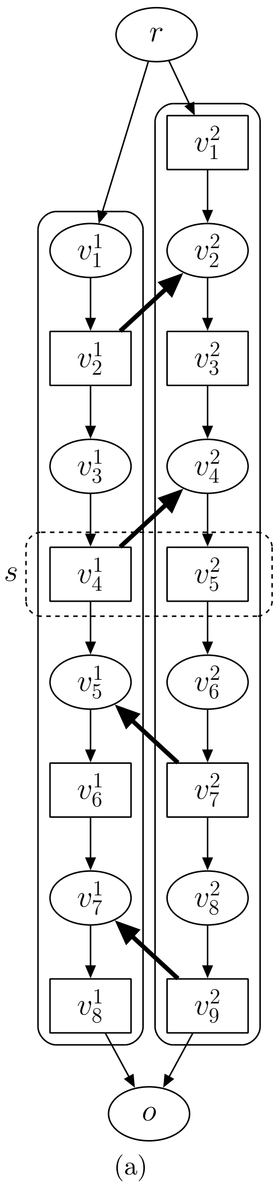
Dispatchers solve such conflicts by forcing the pair of conflicting trains to meet in a specified station, which in turn may not accord with the official timetable. So, for some pair of trains i and j we may force them to meet in a given station s of the railway. I show now how to model the effect of such a decision on the schedule $\boldsymbol{t}$ by adding a suitable set of arcs A_s^{ij} to G^T. Including these in the model prevents any block conflicts, regardless of where the dispatcher decides the trains should meet. The station conflicts however, depend on the assignment of meeting points, and is therefore left to the algorithm. Two strategies for eliminating station conflicts are presented in Section 5.3 on page 37.

In the following, let us distinguish between two cases: i and j travel in opposite directions or they travel in the same direction.

3.5.1 Case 1: Trains moving in opposite directions

Train i and train j, travelling in opposite directions, meet in station s. Clearly, s belongs to both P^i and P^j. So, let v_k^i and v_m^j be the nodes corresponding to station s on P^i and P^j, respectively. Since i and j meet in s, then j leaves s after i has arrived in s, that is $t_{j,m+1} \geq t_{ik}$. Similarly, i leaves s after j enters s, that is $t_{i,k+1} \geq t_{jm}$. This is represented by adding the arcs $A_s^{ij} = \{(v_k^i, v_{m+1}^j), (v_m^j, v_{k+1}^i)\}$ with weight 0 to the graph G^T. This case is illustrated in 3.3b, where A_s^{ij} is shown with bold arcs. Observe that these arcs ensure that i and j will not conflict on a block in the resulting schedule, since trains i and j enter the station from opposite directions (and thus they cannot conflict before they enter) and they exit in opposite directions (and they cannot conflict again after they have met).

As A_s^{ij} depends on the station s, different arcs will be included for different choices of s. Figure 3.4b shows the arcs for all possible choices of s.



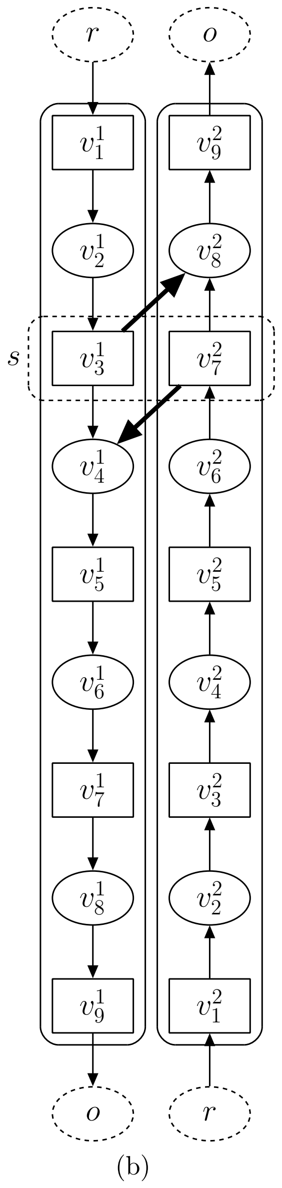


Figure 3.3: Expanding the routes graphs in Figure 3.2 with precedence constraints for meeting trains. The bold arcs represent precedence constraints preventing track conflicts. (a) shows two trains running in the same direction. Train 1 is overtaken by Train 2 at station s. The weight on the bold arcs is zero. (b) shows two trains running in opposite directions, meeting at station s. The source r and sink o have been duplicated in order to allow showing corresponding nodes side by side.

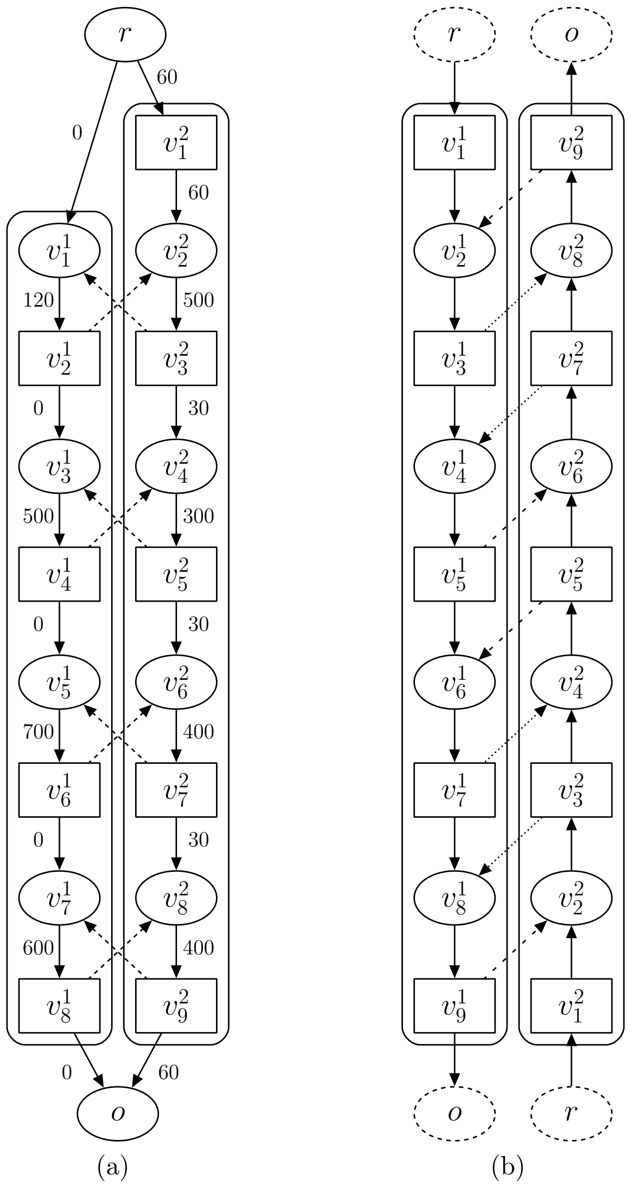

r
o
0
60
120
500
30
300
700
400
600
v_1^1 v_2^1 v_3^1 v_4^1 v_5^1 v_6^1 v_7^1 v_8^1 v_9^1
v_1^2 v_2^2 v_3^2 v_4^2 v_5^2 v_6^2 v_7^2 v_8^2 v_9^2
(a)
(b)

Figure 3.4: Depending on the choice of station where the trains should meet, different arcs must be included. (a) shows two trains running in the same direction. Exactly one arc from each of the dashed or dotted pairs must be included in the graph to prevent a track conflict. The weight on the disjunctive arcs is zero. (b) shows two trains running in opposite directions. Either one of the pairs of dashed or dotted arcs or one of the single dashed arcs at the ends must be included to prevent a track conflict. The source r and sink o have been duplicated in order to allow showing corresponding nodes side by side.

3.5.2 Case 2: Trains moving in the same directions

Train i and train j, travelling in the same direction, meet in station s. This may be necessary if, for example, a train should catch up and overtake another train. This case is a bit more complicated because, for safety reasons, two trains can never be on the same block, even if running in the same direction. So, again let v_k^i and v_m^j be the nodes corresponding to station s on P^i and P^j, respectively. Let us assume that i precedes j before reaching station s, and follows j afterwards. This means that, for every station s' preceding or coinciding with s on P^i, train i must arrive in s' before train j has entered the block which immediately precedes s' on both routes (if such block belongs to P^j). So, if we start considering station s, and assuming that $v_{m-1}^j \in P^j$, then $(v_k^i, v_{m-1}^j) \in A_s^{ij}$. If we now consider the station immediately preceding s on the route P^i, this corresponds to node v_{k-2}^i. Similarly if the block entering such station belongs to the route of j then $v_{m-3}^j \in P^j$, and the arc $(v_{k-2}^i, v_{m-3}^j) \in A_s^{ij}$. Similarly, if $v_{k-4}^i, v_{m-5}^j \in V$, then $(v_{k-4}^i, v_{m-5}^j) \in A_s^{ij}$, and so forth.

The roles of i and j are interchanged after station s, and for every station s' following s on P^j, train j must arrive in s' before train i has entered the block which immediately precedes s' on both routes (if such block belongs to P^i). So, as long as the corresponding nodes are in V, we have that arcs $(v_{m+2}^j, v_{k+1}^i), (v_{m+4}^j, v_{k+3}^i), \ldots$ belong to A_s^{ij}.

The arcs in A_s^{ij} are shown as bold arcs in Figure 3.3a. It is not difficult to see that the inclusion of these arcs will prevent any block conflict for trains i and j in the resulting schedule $\boldsymbol{t}$.

Just as for the opposing trains, all the arcs depending on the choice of s has been included in Figure 3.4a. In each of the four pairs of arcs, exactly one arc must be included.

3.6 Handling track sections

As previously noted, the tracks between stations, as well as the tracks of the stations themselves are often partitioned into several fixed blocks. The extension of the model is straightforward.

As before, a node is associated with each block and each station, but we may now have more than $q-1$ blocks in B, and block i does no longer necessarily lie between stations i and $i+1$.

Since trains travelling in opposite directions are still restricted to meet at stations, Case 1 above is unchanged.

Similarly, a train overtaking another train must still do so at a station, but as there may now be several blocks between stations Case 2 is a little different. Using the same assumptions as above, we see that for every node $v^i_{k-n}, n \in \mathbb{N}_0$ preceding s on P^i where v^i_{k-n-1} does not correspond to a station, train i must enter this node before train j can enter v^j_{m-n-1} (if such block belongs to P^j), adding the arcs

$$\{(v^i_{k-n}, v^j_{m-n-1}) : n \in \mathbb{N}_0, v^i_{k-n} \in P^i, v^j_{m-n-1} \in P^j \setminus S\}$$

Like above, the roles are interchanged after station s, and we must also add the arcs

$$\{(v^j_{m+n+1}, v^i_{k+n}) : n \in \mathbb{N}, v^i_{k+n} \in P^i \setminus S, v^j_{m+n+1} \in P^j\}.$$

Security constraints requiring additional empty blocks between trains are easily accommodated replacing the constant 1's in the above sets of arcs by any number of blocks, while also adjusting the expressions to avoid arcs "crossing" stations.

3.7 Moving blocks

The model used here has no apparent extension to moving blocks. Instead this must be approximated by partitioning the track into short fixed blocks. Koning compares the performance of ETCS Level 2 and Level 3 in [20], and simulates approximately 10 percent smaller headways on free tracks for Level 3 with moving blocks than with Level 2, even for relatively short blocks (600 m). Consequently, even shorter blocks have to be used in order to get closer to the optimal solution, at the cost of a larger model.

Except for in-station movements, the comparisons in [20] consequently shows smaller headway using moving blocks, suggesting that an approximation using the model described here will be feasible for railways using moving blocks, although not optimal.

Chapter 4

Solving optimization problems

This chapter presents a brief introduction to the concepts of mathematical optimization used in this thesis. The reader should be familiar with linear and integer programming and basic graph theory, or study the references presented in the sections below.

4.1 Concepts from graph theory

Graphs have already been used in Section 3.3 to describe the paths of the trains, and the precedence constraints in the model. This section presents some concepts from graph theory that will be used throughout Chapter 5.

4.1.1 Cliques

A *clique* in $G = (V, E)$ is a *completely connected* set of nodes $C \subset V$, i.e., $\{u, v\} \in E$ for all $u, v \in C$. The original definition by Luce and Perry requires a clique not to be a subset of any other clique (and to have at least three nodes)[21], and this definition is still used by some authors. I will instead use the term *maximal clique* to describe a clique that cannot be extended to a larger clique by adding an adjacent node.

A clique is a *maximum clique* if there are no other cliques in the graph with a higher number of nodes. The maximum cliques in Figure 4.1 are $\{A, B, C\}$, $\{A, C, D\}$, $\{C, D, E\}$ and $\{C, D, F\}$. The clique $\{F, G\}$ is maximal but not a maximum clique.

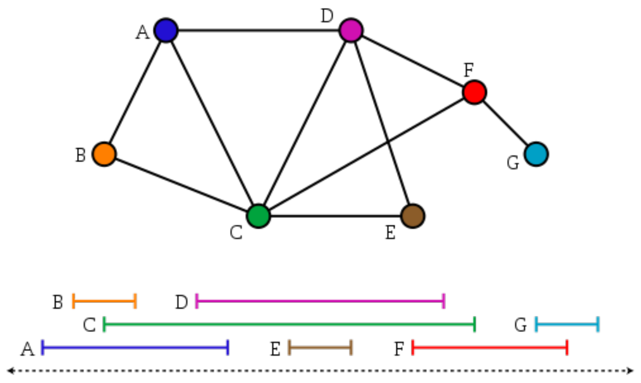


Figure 4.1: An interval graph. The maximal cliques are $\{A, B, C\}$, $\{A, C, D\}$, $\{C, D, E\}$, $\{C, D, F\}$ *and* $\{F, G\}$

4.1.2 Interval graphs

An interval graph is a graph representation of intervals on the real line. Each node represents an interval, and nodes are connected by an edge if and only if their intervals overlap. Figure 4.1 shows an interval graph and the underlying intervals. Note that while the interval graph is uniquely determined by a set of intervals, this does not hold the other way. The graph simply contains information on which intervals that overlap.

Finding cliques in graphs is generally hard. Deciding whether a graph contains a clique larger than a given size is NP-complete [18, pp. 94, 97]. However, the task here is to find cliques in an interval graph, as can be seen in Figure 4.1. Finding the maximal cliques, or determining if a k-clique exists in an interval graph can be done in $O(n \log n)$ time[27]. All one needs to do is to iterate over the intervals, each representing a train, and add the start and end times to a sorted list. Then, using a counter which is initially zero, one starts at the first start time, increasing the counter for every interval that starts and decreasing it for every interval that ends. At any point the value of the counter equals the number of trains in the station.

4.1.3 Flows

The definitions given here correspond to the ones used in [26], where you can also find an extensive treatment of graph theory as used in combinatorial optimization, including flows and circulations.

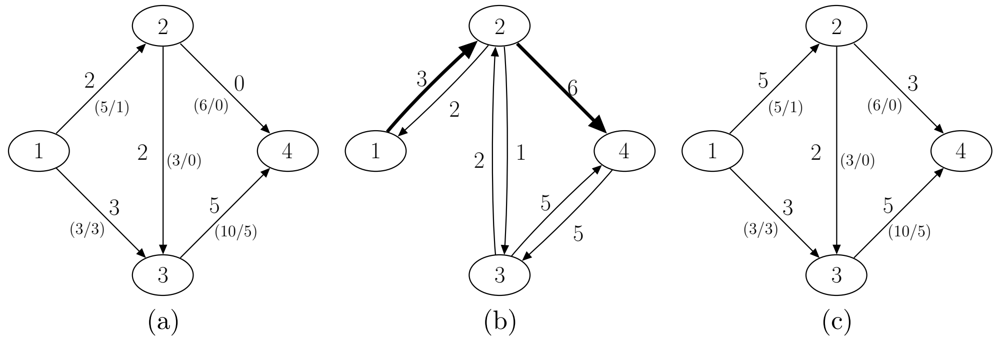


Figure 4.2: Augmenting a flow. (a) An initial flow. The numbers in parantheses are the upper and lower bounds. (b) The residual graph. An augmenting path is emphasized. (c) The new flow. In this case the new flow is maximal.

Given a directed graph $D = (V, A)$ and two nodes $s, t \in V$, an *s-t flow* is a function $x : A \to \mathbb{R}$, such that

$$\begin{array}{llll} i) & x(a) \geq 0 & & \text{for each } a \in A \text{ and} \\ ii) & \displaystyle\sum_{a\in\delta^-(v)} x(a) = \sum_{a\in\delta^+(v)} x(a) & & \text{for each } v \in V \setminus \{s,t\} \end{array}$$

where $\delta^-(v)$ and $\delta^+(v)$ are the sets of arcs leaving and entering v, respectively. In other words, a flow associates a non-negative number with every arc, so that for each node, except a source s and a sink t, the in-flow is equal to the out-flow. Given lower bound l and upper bound f on the arcs, *i)* is replaced by $l(a) \leq x(a) \leq f(a)$. A *circulation* is a flow with no sink or source, i.e., where the in-flow is equal to the out-flow in every node. The *value* of the flow is $\sum_{a\in\delta^-(s)} x(a)$.

Given a graph $D = (V, A)$, with lower bound l and upper bound f and an s-t flow function x, as shown in Figure 4.2a, the *residual graph* is the graph $D_x = (V, A_x)$ in Figure 4.2b where

$$A_x = \{a \mid a \in A,\ x(a) < f(a)\} \cup \{a^{-1} \mid a \in A,\ x(a) > l(a)\}$$

and $a^{-1} = (v, u)$ for $a = (u, v)$. That is, for every arc in A where the flow can be increased A_x contains an equal arc, and for every arc in A where the flow can be decreased A_x contains an opposite arc. An *augmenting path* is an s-t path in D_x. Such a path is emphasized in Figure 4.2b. By adding units of flow along an augmenting path, when such a path exists, we get a new s-t flow with a greater value, shown in Figure 4.2c.

4.2 Linear programming

A linear programming problem or *linear program* is any problem where we seek to maximize or minimize the value of a linear *objective* or *cost* function $f = \sum_i c_i x_i$, subject to linear constraints. It can be written on *standard form* using matrix notation as

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

where $\boldsymbol{x}$ is a vector of n unknowns, and the $m \times n$ constraint matrix A and m-vector b gives the constraint inequalities' left hand coefficients and right hand constants respectively. A thorough treatment of linear programming is given by Vanderbei in [28].

4.2.1 Integer and mixed integer linear programming

An *integer linear programming* (ILP) problem is a linear program with integer unknowns. A *mixed integer linear programming* (MILP) problem has both continuous and integral unknowns. The general ILP or MILP problem is NP-hard [24, pp. 125–126], meaning that no general polynomial-time solution is likely to exist. For certain classes of integer problems though, polynomial-time solutions exist[25].

4.2.2 Piecewise linear programs

Using a piecewise linear objective function gives a more flexible model, and can be used to approximate functions of higher order. Generally, linearly constrained problems having a separable piecewice linear objective function can be converted into a MILP problem, and when we are minimizing (maximizing) over *convex* (concave) piecewise linear functions, as the functions seen in Figure 4.3, the resulting problem is an LP problem[2, pp. 160–161].

This thesis minimizes over monotonically increasing piecewise linear cost functions. The linearization of this kind of objective functions is particularly

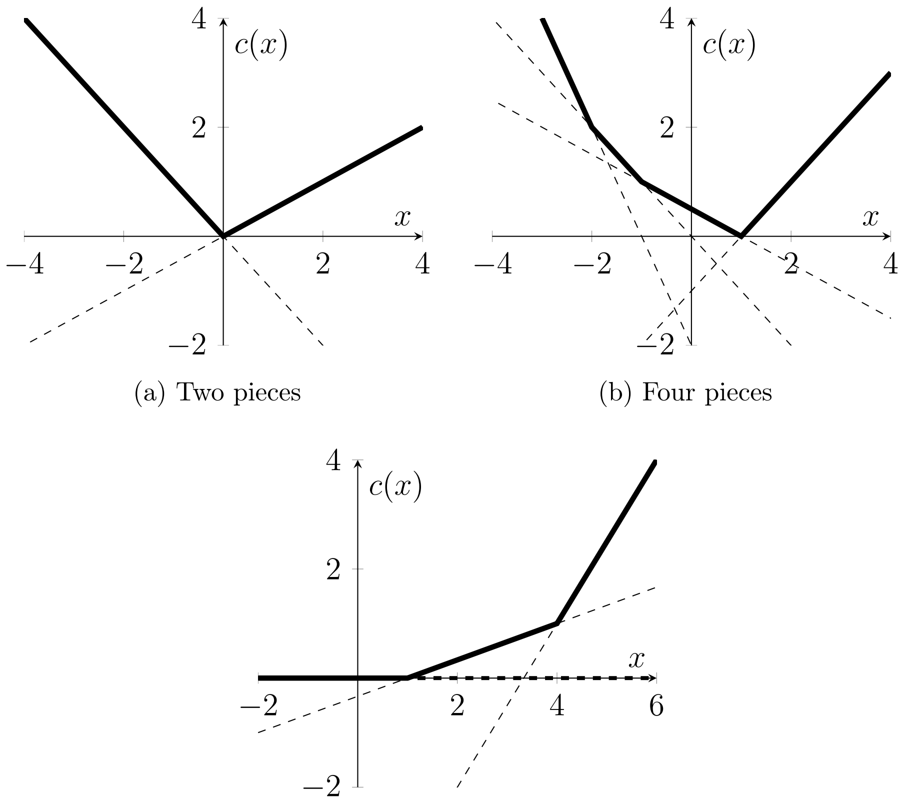


Figure 4.3: Examples of convex piecewise linear function

straightforward. Generally, for n unknowns, we have the problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i(x_i) \\ \text{subject to} \quad & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

where c_i for all $i = 1, \ldots, n$ is a monotonically increasing piecewise linear function, such as the one seen in Figure 4.3c. This can be linearized using one of several methods detailed in [8].

4.3 Solution strategies

For linear problems with continuous unknowns a common solution methods is the *simplex method*, discovered by Dantzig in 1947. It is described in detail in [3]. The simplex method is capable of efficiently solving most linear problems, although exponential time examples for Dantzig‘s algorithm have been constructed[19].

4.4 Separation oracles

A separation oracle for an optimization problem $\min\{f(x) : x \in P\}$, where P is the polyhedron $\{x \in \mathbb{R}^n : Ax \le b\}$ is an algorithm that takes as input an n-vector $\bar{x}$ and either determines that $\bar{x} \in P$ or returns an inequality $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \le \boldsymbol{b}$ that is valid for all solutions in P, but violated by $\bar{x}$.[1]

4.5 The cutting plane method

A cutting plane method combines a separation oracle and a linear program solver. The dynamic simplex method combines a separation oracle and the simplex method to solve optimization problems that have an impractically large number of constraints. Even when it is possible to list or compute all the constraints, there are a lot of cases where computing and including all the constraints may be too time and memory consuming, or there may be a large number of constraints that slow down the optimization.

First, one relaxes the original problem, e.g. by dropping the integrality constraints, so that one has an LP problem solvable by the simplex method. Then the simplex method is applied in order to find an optimal solution for the relaxed problem. As long as the solution violates any of the constraints in the original problem, one or more of the violated constraints are added to the relaxed problem, and the simplex method is reapplied. An optimal solution of the relaxed problem is a lower bound for the original problem, so when an optimal solution of the relaxed problem also lies in the the original solution space, it must be an optimal solution also for the original problem.

Chapter 5

Algorithm

This chapter describes the formulations and algorithms used to solve the Single-track Traffic Control Problem to optimality.

5.1 Evaluating the actual timetable

The quality of the *actual timetable* depends on its conformity to the *official timetable*. For each train $i \in T$ and each station s represented in P^i, let v let $OA_v = OA^i_s$ and $OD_u = OD^i_s$ be the official arrival and departure time, respectively. Observe that if s corresponds to node $v^i_k \in P^i$, then the actual arrival time in s is $t_v = t_{i,k}$, whereas the actual departure time is $t_{i,k+1}$. Deviating from the official timetable is costly, so we are given a convex cost function c_v for each $v \in V$, and the total cost is given by

$$c(\boldsymbol{t}) = \sum_{v \in V} c_v(t_v) = \sum_{v \in V^i \forall i \in T} c_v t_v = \sum_{v^i_k \in P^i \forall i \in T} .$$

In this thesis, only delayed arrivals will be assigned a cost. Departing early will be disallowed, and departing late has zero cost in itself, though it may lead to a costly delayed arrival at the next stop. The cost function can then be expressed as

$$c(\boldsymbol{t}) = \sum_{v \in V : rr(v) \in S} c_v(t_v - OA^{tr(v)}_{rr(v)}).$$

In my simulations I have used the same cost function for all nodes, namely the piecewise linear function shown in Figure 5.1, with breaking points at 0, 3, 5 and 10 minutes.

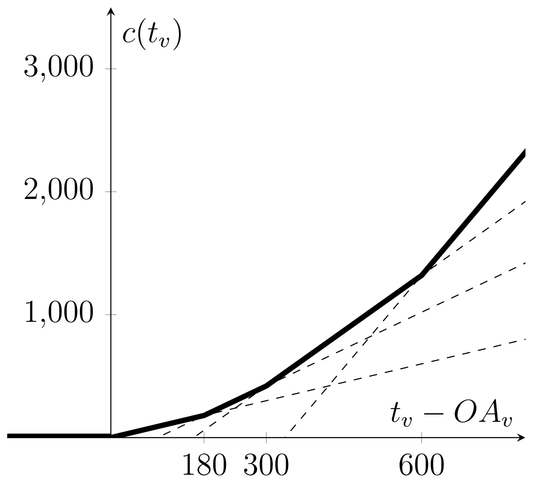


Figure 5.1: The function used for the cost of arriving late. The time is given in seconds. The cost per second of a delay is given by the slopes of the function and is 1 up to three minutes, then 2 up to 5 minutes, 3 up to 10 minutes and then 5. Arriving early is costless.

5.2 A MILP formulation for STC

The algorithm needs to identify and resolve possible conflicts. In the following the set of possibly conflicting pairs of trains will be denoted by $K = \{\{i, j\} : i \in T, j \in T, i \text{ and } j \text{ conflicting}\}$. How to determine this set will be detailed in Section 5.4 on page 42. To simplify the notation let $ij = \{i, j\}$. For every pair of possibly conflicting trains $ij \in K$, let $S(ij)$ be the set of stations where i and j can actually meet. For every $ij \in K$, $s \in S(ij)$, let $y_s^{ij} = 1$ if i and j meet in s, and 0 otherwise. Denote by $G(\boldsymbol{y})$ the graph obtained from G^T by including the arcs of A_s^{ij} when $y_s^{ij} = 1$, for all $ij \in K$, $s \in S$. Let $\boldsymbol{t}(\boldsymbol{y})$ be the schedule computed on $G(\boldsymbol{y})$. Then the Single-track Traffic Control problem (STC) amounts to finding a binary vector $\boldsymbol{y}$ such that $\boldsymbol{t}(\boldsymbol{y})$ is conflict free and $c(\boldsymbol{t}(\boldsymbol{y}))$ is minimized.

The following is an integer programming formulation for the STC problem:

$$\begin{array}{lll} \min & \sum_{v \in V} c_v(t_v) & \\ s.t. & & \\ (i) & t_v - t_u \geq W_{uv}, & (u,v) \in A \\ (ii) & t_v - t_u \geq M(y_s^{ij} - 1), & ij \in K, s \in S(ij), (u,v) \in A_s^{ij} \\ (iii) & \boldsymbol{t} \text{ conflict free}, & \boldsymbol{t} \in \mathbb{R}^V, y_s^{ij} \in \{0,1\}, ij \in K, s \in S(ij) \end{array} \qquad (5.1)$$

where M is a large suitable constant.

Next, I show how to represent constraint (*iii*) by introducing suitable variables and linear inequalities. Note that we only need to ensure that $\boldsymbol{t}(\boldsymbol{y})$ does not imply station conflicts, since block conflicts are prevented by the additional arcs (constraints (*ii*)).

5.3 Strategies for eliminating station conflicts

5.3.1 Strategy 1: Cover cuts

The actual timetable can be regarded as a set of interval graphs, one for each station. Using the schedule $\boldsymbol{t}$, the intervals for each train begins when the train enters the station, and ends when it leaves. For any two trains i and j in T, their intervals in station $s \in S$ overlap if and only if $y_s^{ij} = 1$. Finding station conflicts in a given station s amounts to finding a set of trains $C \in T$ with cardinality $|C| > d_s$ such that $y_s^{ij} = 1$ for all $i, j \in C, i \neq j$, i.e., a clique in the graph $G = (T, Y_s)$ where $Y_s = \{(i,j) : y_s^{ij} = 1 \;\forall\; i, j \in T, i \neq j\}$ for any $s \in S$.

As mentioned in Section 4.1.2 on page 30, deciding whether such a clique exists, can be done in $O(n \log n)$ time, by ordering the intervals as shown in Figure 5.2 and iterating over them from first to last. This way we can generate all the maximal cliques at the same time. However, we are only interested in the cliques with exactly $d_s + 1$ trains, as ruling them out would also rule out larger cliques at the same time. This is implemented here by iterating over the intervals while keeping a list of the latest arrivals, removing a train from the list either when it departs, or when the list has exactly $d_s + 1$ trains and it is the train in the list that departs first. This does not generate all $\binom{n}{d_s}$ sub-cliques of a maximal n-clique, but it generates a set of cliques covering the maximal clique, and does so in $O(n \log n)$ time. For all the violating cliques generated, the cover cut $\sum_{i,j \in C} y_s^{ij} \leq \frac{|C|(|C|-1)}{2}$ is added to the formulation.

5.3.2 Strategy 2: A compact, flow based representation of station conflict-free solutions

Let us first fix a meet assignment $\bar{y}$. For any train $j \in T$, let $Su(j, s, \bar{y})$ be the set of *successors* of j in station s, that is the set of trains $i \in T$ which enter

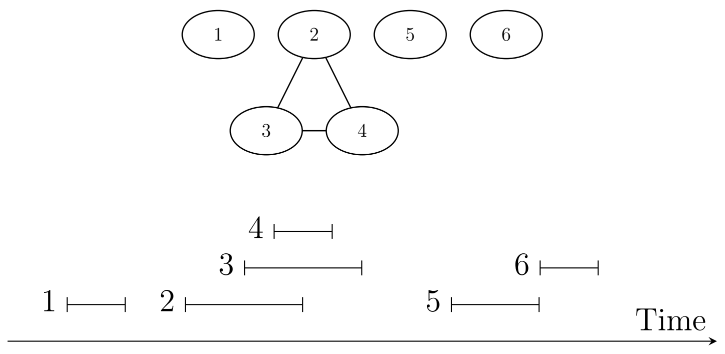


Figure 5.2: Train stop intervals at a station, and the corresponding interval graph. The cliques with size greater than one are $\{2, 3\}$, $\{2, 3, 4\}$ *and* $\{3, 4\}$. *If the station capacity is less than three, constraints must be added in order to avoid a station conflict.*

s after j leaves the station. Note that since the meet assignment is given, $Su(j, s, \bar{y})$ is known for all $j \in T$ and $s \in S$ (if s is visited by j). Now, think of station platforms as unit resources that can be supplied to trains. Then a train i receives the platform either "directly" from the station s, or from a train j such that $i \in Su(j, s, \bar{y})$, which received the platform at an earlier stage. Then the assignment $\bar{y}$ is station conflict-free if every train receives the required platform, as I will show more formally below. This feasibility problem can be represented as a network flow problem, where nodes are associated to the station s and to the trains. Station s is a supply node (it supplies up to d_s units of resource) and every train j can act both as a demand node and as a supply node, since it can supply 1 unit of resource to successive trains.

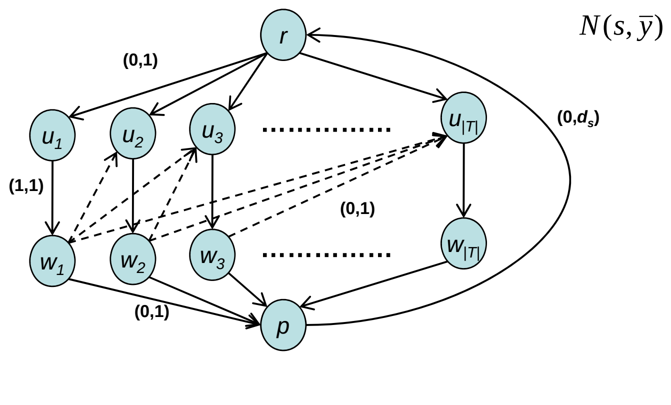


Figure 5.3: The support network. The possible arcs for E_W *are dashed.*

Now, concentrate on a given station s. To simplify the notation, assume that every train in T will go through s. Since both s and $\bar{y}$ are fixed, let $Su(j, s, \bar{y}) = Su(j)$. Also, assume that trains are ordered by their arrival times in station s. So, $j \in Su(i)$ implies $j > i$.

Let us introduce the support graph $N(s, \bar{y}) = (\{r, p\} \cup U \cup W, E)$ shown in Figure 5.3, where $U = \{u_1, \ldots, u_{|T|}\}$, $W = \{w_1, \ldots, w_{|T|}\}$. Let the arc set be defined by

$$\begin{aligned}
E &= E_r \cup E_U \cup E_W \cup E_p \cup \{(p, r)\} \\
E_r &= \{(r, u) : u \in U\} \\
E_U &= \{(u_j, w_j) : j \in T\} \\
E_W &= \{(w_i, u_j) : j \in Su(i)\} \\
E_p &= \{(w, p) : w \in W\}
\end{aligned}$$

With each arc $e \in E$ associate lower bound l_e and upper bound f_e. Namely, $l_e = 0$ and $f_e = 1$ for $e \in E_r \cup E_W \cup E_p$. Then $l_e = f_e = 1$ for $e \in E_U$ and finally $l_{pr} = 0$, $f_{pr} = d_s$.

Theorem The assignment $\bar{y}$ is station conflict-free if and only if, for every $s \in S$, the graph $N(s, \bar{y})$ has a circulation satisfying all lower and upper bounds.

Proof Given a station conflict-free assignment $\bar{y}$, I will show that a circulation on $N(s, \bar{y})$ exists. For a given station s, consider the graph N' obtained from $N(s, \bar{y})$ by removing the arc (p, r), and let $\boldsymbol{z}$ be an r-p flow on N'. Such a flow exists regardless of $\bar{y}$, as we can let

$$z_e = \begin{cases} 1 & \text{if } e \in E_r \cup E_U \cup E_p \\ 0 & \text{if } e \in E_W \end{cases}$$

Now, let z_e be a minimum r-p flow satisfying the upper and lower bounds. Then, until there are no pairs of trains $j, k \in Su(i), j < k$ where $i \in T$ and $z_{(w_i, u_k)} = z_{(r, u_j)} = 1$ we add one unit of flow along the cycle

$$r, (r, u_k), u_k, (u_k, w_i), w_i, (w_i, u_j), u_j, (u_j, r), r$$

in the residual graph. As this does not change the value of the flow, it is still a minimum r-p flow.

We now have that for every train $j \in T$ such that $z_{(r, u_j)} = 1$ either

1. $j \notin Su(i)$ for all $i \in T, i < j$,
2. $z_{(w_i,u_k)} = 0$ for all $i, k \in T$ such that $j, k \in Su(i)$ and $k > j$.

This in turn implies that for a train $j \in T$ such that $z_{(r,u_j)} = 1$, there are already $\sum_{i \in T, i<j} z_{(r,u_i)}$ trains in the station when j enters it. Finally, as $\bar{y}$ is a station conflict-gree assignment, $\sum_{i \in T, i<j} z_{(r,u_i)} < d_s$ where j is the last train such that $z_{(r,u_j)} = 1$, which in turn proves that $\sum_{e \in E_r} z_e \leq d_s$. By applying this flow to the original graph $N(s, \bar{y})$ and adding $\sum_{e \in E_r} z_e$ units of flow on the edge (p, r), we obtain a circulation satisfying all lower and upper bounds.

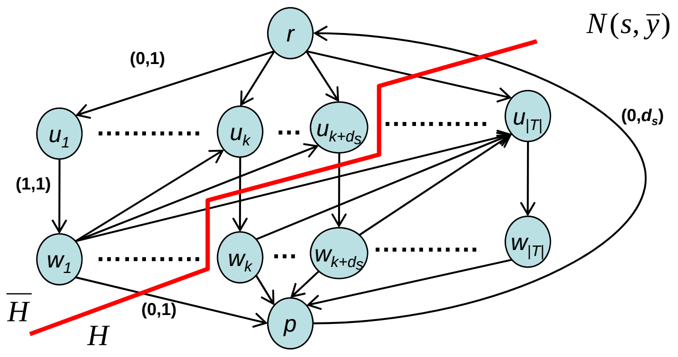


Figure 5.4: The cut in the necessity proof.

Now it remains to show that if there does not exist a platform assignment for some station s, then $N(s, \bar{y})$ does not admit a circulation. Hoffman's circulation theorem[26, p. 171] states that N does not have a circulation if and only if there exists a set of nodes H such that $\sum_{e \in \delta^-(H)} l_e > \sum_{e \in \delta^+(H)} f_e$. So, assume that a platform assignment does not exist, then there exist $d_s + 1$ trains, say $Q = \{k, k+1, \ldots, k+d_s\} \subseteq T$, which are simultaneously in station s. We construct a cut by letting

$$H = \{p\} \cup \{w_j : j \in \{k, \ldots, |T|\}\} \cup \{u_j : j \in \{k + d_s + 1, \ldots, |T|\}\}.$$

We then have $\sum_{e \in \delta^-(H)} l_e = |Q| = d_s + 1$ since the only arcs with positive lower bound (the arcs in E_U) entering H are precisely the arcs $(u_k, w_k), \ldots, (u_{k+d_s}, w_{k+d_s})$. All other arcs in E_U are either completely contained in H, for $j > k + d_s$, or in the complement $\overline{H}$ of H, for $j < k$.

On the other hand, it is easy to see that the only arc going out from H is $\delta^+(H) = \{(p, r)\}$, which implies $\sum_{e \in \delta^+(H)} f_e = d_s < c_s + 1 = \sum_{e \in \delta^-(H)} l_e$. In fact:

1. $E_r \cap \delta^+(H) = \emptyset$. Indeed, all arcs in E_r are outgoing from r and $r \in \overline{H}$.

2. $E_U \cap \delta^+(H) = \emptyset$. Indeed, $u_j \in H$ for $j = k + d_s, \ldots, |T|$. But then also $w_j \in H$ for $j = k + d_s, \ldots, |T|$.

3. $E_W \cap \delta^+(H) = \emptyset$. We must show that $(w_i, u_j) \notin E_W$ for $i \geq k$ and $j \leq k + d_s$. That is, we show that for $j \in \{1, \ldots, k, \ldots, k + d_s\}$ and $i \geq k$, then $j \notin Su(i)$. This is trivial for $i > k + d_s$ since $j \notin Su(i)$ for all $i > j$. Also, by assumption, the trains in $Q = \{k, \ldots, k + d_s\}$ are simultaneously in the station, which implies that $j \notin Su(i)$ for all $j, i \in Q$.

4. $E_p \cap \delta^+(H) = \emptyset$. Trivial, since $p \in H$ and all arcs in E_p are incoming in p.

This concludes the proof.

The above result can be used to model the station capacity constraint into the MILP program. To this end, introduce a binary variable x_s^{ij} for all stations $s \in S$ and all pairs of trains $i \in T, j \in T$, with the interpretation that $x_s^{ij} = 1$ if and only if $j \in Su(i, s, y)$. Observe that, if i and j are not conflicting trains, then the value of x_s^{ij} can be easily derived from the official timetable or from the current status of the trains. Otherwise, $\boldsymbol{x}$ is a function of $\boldsymbol{y}$ and it can be easily expressed by linear constraints.

Then we need to represent, for each station $s \in S$ the network flow problem discussed above on the graph $N(s, y)$. This can be done by considering an extended flow network $\overline{N}$ obtained from N by letting $E_W = \{(w_i, u_j) : i \in T, j \in T\}$, leaving all other arc sets unchanged. So, E_W contains all possible arcs from W to U. Observe that $\overline{N}$ is independent of $\boldsymbol{y}$. However, to prevent sending flow on "forbidden" arcs, fix the upper bound $f_{w_i, u_j} = 0$ whenever $j \notin Su(i)$ (this in turn depends on $\boldsymbol{y}$).

Next, for every arc $e \in \overline{N}$, introduce a flow variable z_s^e, with lower and upper bounds $l_e \leq z_s^e \leq f_e$. Then impose on each node $v \in \overline{N}$ the flow conservation constraints $\sum_{e \in \delta^-(v)} z^e = \sum_{e \in \delta^+(v)} z^e$.

Lower and upper bounds are defined as for $N(s, y)$ except for the arcs in E_W. For such arcs simply let $f_{w_i u_j}^s \leq x_s^{ij}$. In this way, the arc (w_i, u_j) can be used to send one unit of flow only if $j \in Su(i, s, y)$.

To see how $\boldsymbol{x}$ and $\boldsymbol{y}$ are related, consider two conflicting trains i, j and a station s. In what follows I write $s_1 \prec_{ij} s_2$ ($s_1 \succ_{ij} s_2$) if station s_1 precedes (follows) stations s_2 on the route of i and j.

Assume i and j travel in the same direction and let i be the initial follower. Let $\bar{s}$ be the meeting station, i.e., $y^{ij}_{\bar{s}} = 1$. Denote by $\overline{S} = \{s \in S : s \prec_{ij} \bar{s}\}$. Then $s \in \overline{S}$ implies $x^{ji}_s = 1$ (i follows j in s, thus $i \in Su(j, s, y)$). For all remaining stations but $\bar{s}$ we have $x^{ij}_s = 1$ (j follows i in s). This can be expressed by the following set of linear constraints:

$$x^{ji}_s = \sum_{q \succ_{ij} s} y^{ij}_q, \qquad s \in S.$$

Then, we write for all $s \neq \bar{s}$,

$$x^{ij}_s = 1 - x^{ji}_s, \qquad s \in S \setminus \{\bar{s}\}.$$

Finally, let $x^{ij}_{\bar{s}} = 0$.

5.4 Identifying possible conflicts

In the foregoing I have used the set $K = \{\{i, j\} : i \in T, j \in T, i \text{ and } j \text{ conflicting}\}$. The naive approach is to include all pairs of trains in K. That ensures that all possible conflicts are covered, but introduces unnecessarily many variables and constraints. I will instead use a small set of pairs of possibly conflicting trains, and dynamically add new pairs if necessary. Initially let the set K be the pairs of trains that are scheduled to meet or directly follow each other in any station. Alternately, K can initially contain just the pairs of crossing trains, or no trains at all, causing more conflicting pairs to be discovered and added dynamically.

At any point in the dynamic or branch and cut search, we test for pairs of trains having conflicting schedules, and add them to the formulation, along with corresponding sets of disjunctive arcs.

Chapter 6

Results

The purpose of this chapter is to see if the algorithm is capable of returning an optimal solution for the single-track traffic control problem within the short time frame available in a real-time setting. The algorithm is tested using three different test inputs, presented below. While the small and large instances are artificial, the medium-sized instance is a real snapshot from the Trento – Bassano line in Italy.

The algorithm was implemented in two versions, one for each of the conflict elimination strategies in Section 5.3. Both implementations are in C++ using the Concert Technology API of the IBM ILOG CPLEX Optimizer. The following are the results of running the two versions on three test cases with an increasing number of trains and stations. All tests were run on the same computer, a workstation with a 2.93 GHz quad-core hyper-threaded Intel® Core$^{\text{TM}}$ i7 CPU, under Red Hat Enterprise Linux (64-bit).

The results from the different instances are presented below, along with a brief summation of the differences between the results. Further discussion is saved for Chapter 7.

6.1 Small instance

This instance has 6 trains and 10 stations. All trains travel the whole line, with half the trains running in each direction. While the input is small, all trains have an initial delay causing conflicts between their schedules.

The results for both conflict separation strategies are shown in Table 6.1. The total number of rows, columns and non-zero elements show the dimension and sparsity of the constraint matrix, illustrating the size of the problem. The number of binary variables is the number of variables that the program must fix to zero or one. For a (continuous) linear program this number would be zero.

CPLEX starts by performing a presolve routine, reducing the number of rows and columns. The number of remaining rows, columns, non-zeros and binary variables show the size of the problem that must be solved in the search phase.

The search methods are chosen automatically by CPLEX, and differ between the two conflict strategies. Also, as shown in Section 6.5, the gain from multi-threading differs between the versions, and consequently the number of threads used differ.

The numbers of different cuts applied is also shown, and shows how many cuts are needed to strengthen the relaxation of the initial formulation.

Both versions present an optimal solution. The solution value gives the minimum total cost of all delays for all trains in the data set. As shown in Chapter 5, the cost is simply an increasing function of the delay in time. It shows the extent of delays, but is not directly a financial cost. Finally, the time taken to obtain an optimal solution is listed.

	Conflict strategy	
	Cover cuts	Resource flow
Total number of rows	902	1770
Total number of columns	421	1281
Total number of non-zero elements	2461	4952
Total number of binaries	180	538
Presolve time (seconds)	0.02	0.02
Reduced MIP rows	979	1405
Reduced MIP columns	332	751
Reduced MIP nonzeros	1696	4193
Reduced MIP binaries	152	247
MIP search method	Branch & cut	Dyn. search
Parallel mode	no	yes
Implied bound cuts applied	102	850
Mixed integer rounding cuts applied	-	146
Gomory fractional cuts applied	12	55
Total cuts applied	114	1051
Solution value	9810	9810
Optimal	yes	yes
Solution time (seconds)	0.48	2.32

Table 6.1: Detailed results from running the small instance

6.2 Medium-sized instance

This instance uses real data collected from the Trento – Bassano line in Italy. It has 23 stations, including simple stops, with capacities varying from 1 to 3 trains. Out of the total of 31 trains, 21 run the whole line in either direction, while 10 trains traffic only the first 13 stations. 3 of the trains are already underway at the start of the simulation. The results for both conflict separation strategies are shown in Table 6.2.

The size of this problem is obviously a lot larger, resulting in a larger constraint matrix. Compared with the small instance, though, a lot more rows and columns are eliminated from the initial problem during the presolve routine.

As the problem is larger, more cuts are needed, and the solution time is longer. While the solution time is very much longer for the resource flow strategy, the increase is less for the cover cut strategy.

	Conflict strategy	
	Cover cuts	Resource flow
Total number of rows	20375	64636
Total number of columns	13931	59280
Total number of non-zero elements	49560	167242
Total number of binaries	11578	33313
Presolve time (seconds)	0.08	0.37
Reduced MIP rows	12211	17783
Reduced MIP columns	4121	18748
Reduced MIP nonzeros	35065	73608
Reduced MIP binaries	2357	3293
MIP search method	Branch & cut	Dyn. search
Parallel mode	no	yes
Cover cuts applied	-	1
Implied bound cuts applied	113	628
Mixed integer rounding cuts applied	-	2699
Gomory fractional cuts applied	71	289
Total cuts applied	184	3617
Solution value	1355	1355
Optimal	yes	yes
Solution time (seconds)	1.46	17.84

Table 6.2: Detailed results from running the medium-sized instance

6.3 Large instance

In order to test how the algorithm reacts to an increased number of trains, this instance was created with twice the number of trains as the medium-sized instance, but with the same number of stations, i.e., 62 trains and 23 stations. As with the medium-sized instance, approximately two thirds of the trains run the whole line. The results for both conflict separation strategies are shown in Table 6.3.

	Conflict strategy	
	Cover cuts	Resource flow
Total number of rows	64797	240701
Total number of columns	51886	232240
Total number of non-zero elements	144287	600811
Total number of binaries	47181	133896
Presolve time (seconds)	0.33	2.76
Reduced MIP rows	24422	36652
Reduced MIP columns	8242	60729
Reduced MIP nonzeros	70130	193683
Reduced MIP binaries	4714	6586
MIP search method	Branch & cut	Dyn. search
Parallel mode	no	yes
Implied bound cuts applied	217	2642
Flow cuts applied	-	1003
Mixed integer rounding cuts applied	-	974
Gomory fractional cuts applied	99	350
Total cuts applied	316	4969
Solution value	1595	1595
Optimal	yes	yes
Solution time (seconds)	5.27	45.07

Table 6.3: Detailed results from running the large instance

6.4 Comparing the sizes of the instances

Figures 6.1–6.4 graph the dimensions of the constraint matrices in the three instances before and after presolving, as listed in Tables 6.1–6.3. In Figures 6.1 and 6.2 the horizontal axis shows the squared product of the number of trains and stations, while in Figures 6.3 and 6.4 the product is not squared.

While, for both conflict strategies, the number of rows, columns, non-zeros and binaries in the initial problem is approximately proportional to the squared product of the number of trains and stations, the corresponding numbers in the reduced problems are approximately proportional to the product, not the squared product, of the number of trains and stations. The exceptions are the number of non-zeros and columns in the reduced problem for the flow formulation, which are approximately proportional to $(|S| \cdot |T|)^{3/2}$.

6.5 Impact of multi-threading

In order to see the parallelization capabilities of the two strategies, a comparison was done using the medium-sized instance and different number of threads. The results are shown in Table 6.4. It is clear that there is little or no impact of multi-threading when using the cover cut strategy, while the resource flow strategy clearly benefits from parallelization.

Threads	**Cover cuts**	**Resource flow**
1	1.46	25.39
2	1.48	22.49
4	1.40	18.66
8	1.46	17.84

Table 6.4: Running time comparison with different number of threads. Times in seconds.

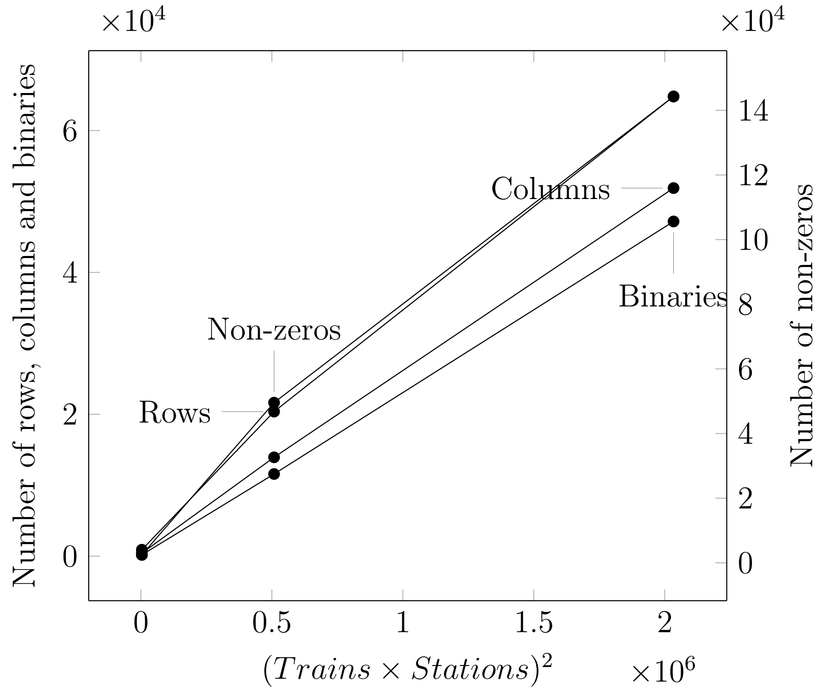


Figure 6.1: Dimensions of the constraint matrix for the cover cut formulation

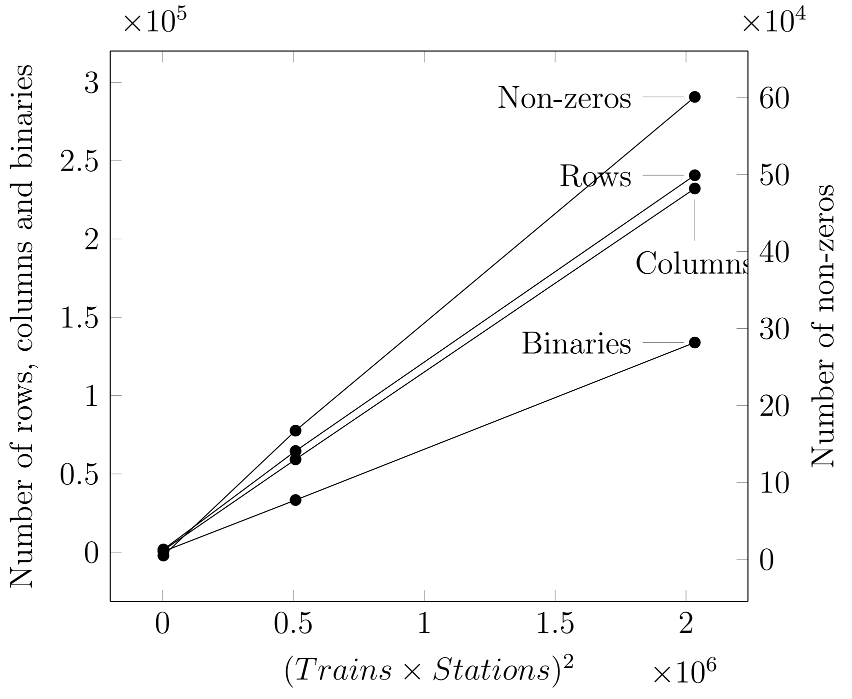


Figure 6.2: Dimensions of the constraint matrix for the flow formulation

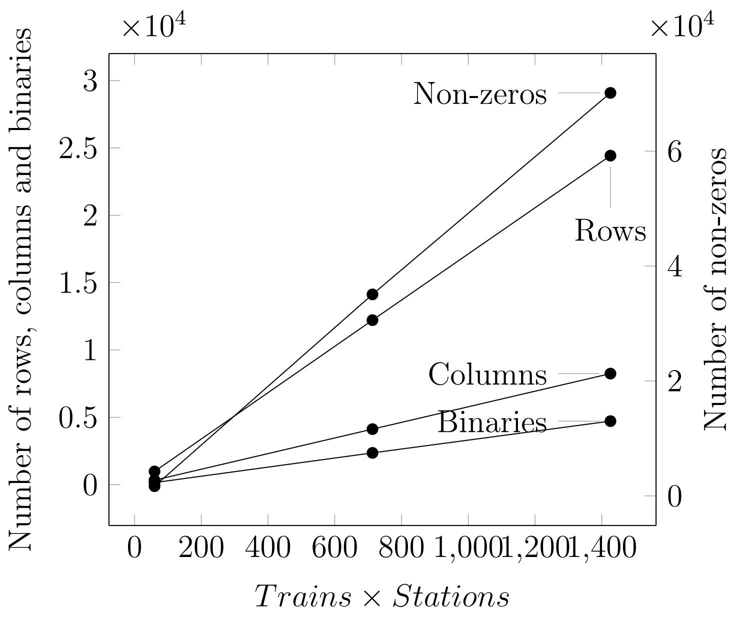


Figure 6.3: Dimensions of the reduced constraint matrix for the cover cut formulation

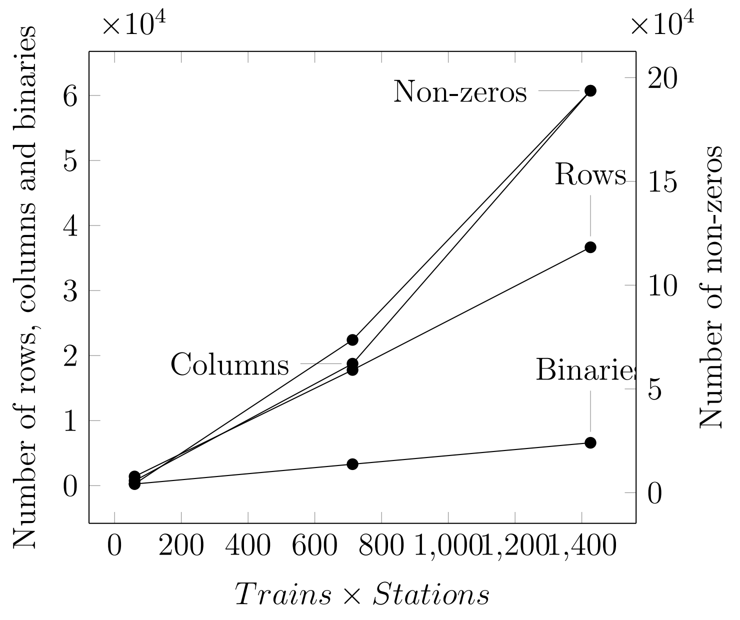


Figure 6.4: Dimensions of the reduced constraint matrix for the flow formulation

Chapter 7

Discussion and Conclusion

In this final chapter, I will discuss the results found in Chapter 6. In order to be of practical use, a real-time traffic control optimizer must be able to solve the STC to (near) optimality within seconds. In [20], Koning puts the total system response time for ETCS Level 2 at 19 seconds and for ETCS Level 3 with moving blocks at 31 seconds. This is the time from automatic position reports are collected to the updated movement authority is displayed to the driver, and gives us a reference frame for the running time of the rescheduling iterations. In practice the response time for Level 2 will be longer if the sensors on the track are sparse.

7.1 Comparing the two strategies for conflict-free solutions

Tables 6.1–6.3 and Figures 6.1–6.4 show that the flow formulation strategy generates vastly larger coefficient matrices than the cover cut strategy, even though they both grow at a rate proportional to the squared product of the number of trains and stations. The larger parallelization benefits from using the flow formulation strategy does not make up for this difference. It is also worth noting that the size of the coefficient matrix after CPLEX has done presolving grows faster for the flow formulation strategy, which implies that the search space is increased accordingly.

As the results show, the implementation using cover cuts performs significantly better than the implementation using the flow formulation, for all input sizes. It is not surprising that the separation using flow formulation has worse scaling properties, as the number of variables is higher. Since the cover cut strategy is also more flexible, as the separation oracle can easily be extended to consider new rules or even special prohibited cases, it is a better choice for real world implementations.

7.2 General performance of the algorithm

The results show that the better of the two strategies performs within an acceptable time frame when applied to real world data. When it is fed with much larger instances, the running time grows larger than the wanted time frame, but this is to be expected. As this implementation only supports single-track lines, extensions and partitions must already be considered when applying it to more complex networks, e.g. where several single-track lines connect at a station. In such cases, simply partitioning the problem spatially will give faster but normally not optimal results.

The CPLEX parameters have not been systematically explored, and along with a dedicated strategy for branching, it would be reasonable to expect a significant reduction of the running time.

7.3 Implementation challenges

A real railway network has complex features that either have to be assumed irrelevant or incorporated into the model. As the model used here is very simple and extendable, such features can be included when building the routes graph. The challenge lies in finding a reasonable balance between complete details and complexity, so that all relevant train routes and schedules are allowed, while keeping computation time and memory usage at acceptable levels.

In order to calculate optimal schedules with high precision, it is important to have reliable and precise input data. If we do not know how soon a train can reach the next station, we will have to extrapolate based on the latest information. All such uncertainty in the input increases the uncertainty in the output, and increases the risk that the optimal schedule found is either infeasible or not really optimal.

7.4 Conclusion and future research

My conclusion is that the model used herein combined with the algorithm presented, using the cover cuts strategy, gives optimal solutions to the single-track traffic control problem within a satisfactory running time in a real-time setting.

Future developments of the algorithm should establish a branching scheme for the meet assignments (the y-variables), in order to achieve faster convergence.

Implementing an algorithm for finding an initial valid, but not optimal, integer solution could also decrease the time needed to find an optimal integer solution.

Without good input data, it is difficult for both man and machine to calculate optimal schedules in real-time. Consequently, is is adamant that modern train control systems are widely implemented. Knowing the exact positions of the trains, along with the condition of the infrastructure, will also benefit the customers through better real-time public information.

References

[1] Geir Dahl and Carlo Mannino. *Notes on Combinatorial Optimization.* Published online, 2011. URL: `http://heim.ifi.uio.no/~geird/comb_notes.pdf`.

[2] George Bernard Dantzig and Mukund Narain Thapa. *Linear Programming: Introduction.* Springer series in operations research. New York, USA: Springer, 1997. ISBN: 0-387-94833-3.

[3] George Bernard Dantzig and Mukund Narain Thapa. *Linear Programming: Theory and extensions.* Springer series in operations research. New York, USA: Springer, 2003. ISBN: 0-387-98613-8.

[4] Andrea D'Ariano. "Improving real-time train dispatching: models, algorithms and applications". In: NGInfra PhD Thesis Series on Infrastructure 18 (2008).

[5] European Commission, Energy and Transport. *ERTMS – Delivering Flexible and Reliable Rail Traffic.* [Online; accessed 2011-09-24]. Brussels, Belgium, 2006. URL: `http://bookshop.europa.eu/en/ertms-delivering-flexible-and-reliable-rail-traffic-pbKO7205273/`.

[6] European Commission (Press release). *Commission facilitates interoperability for Europe's trains.* [Online; accessed 2011-09-24]. 2008-04-23. URL: `http://europa.eu/rapid/pressReleasesAction.do?reference=IP/08/629\&format=HTML\&aged=1\&language=en\&guiLanguage=en`.

[7] European Commission (Press release). *European rail transport: a major step towards a harmonised signalling system.* [Online; accessed 2011-09-24]. 2009-07-22. URL: `http://europa.eu/rapid/pressReleasesAction.do?reference=IP/09/1167\&format=HTML\&aged=0\&language=en\&guiLanguage=en`.

[8] James K. Ho. “Relationships among linear formulations of separable convex piecewise linear programs”. In: *Mathematical Programming: Essays in Honor of George B. Dantzig Part I*. Ed. by Richard W. Cottle et al. Vol. 24. Mathematical Programming Studies. Springer Berlin Heidelberg, 1985, pp. 126–140. ISBN: 978-3-642-00919-8.

[9] *IEEE Standard for Communications-Based Train Control (CBTC) Performance and Functional Requirements*. IEEE Standard 1474.1. The Institute of ElectricalElectronics Engineers, Inc., 2004. URL: `http://ieeexplore.ieee.org/servlet/opac?punumber=9643`.

[10] International Union of Railways. *ETCS Implementation Handbook*. [Published online]. 2008. URL: `http://www.uic.org/IMG/pdf/etcs_handbookf.pdf`.

[11] International Union of Railways. *What is ERTMS?* [Online; accessed 2011-09-24]. 2011-03-29. URL: `http://www.uic.org/spip.php?article381`.

[12] Jernbaneverket (Norwegian National Rail Administration). *Godstransport på bane, Jernbaneverkets strategi*. [Online; accessed 2011-09-24]. 2007. URL: `http://www.jernbaneverket.no/no/dokumenter/Prosjekter/Godstransport-pa-bane---Jerbaneverkets-strategi/`.

[13] Jernbaneverket (Norwegian National Rail Administration). *Grafiske togruter f.o.m 12.juni 2011*. [Online; accessed 2011-08-21]. 20.05.2011. URL: `http://www.jernbaneverket.no/no/Marked/Informasjon-for-togselskapa/Grafiske-togruter-fom-12juni-2011/`.

[14] Jernbaneverket (Norwegian National Rail Administration). *Jernbanestatistikk 2008 / Railway Statistics 2008*. Tech. rep. Oslo, 2008. URL: `http://www.jernbaneverket.no/no/dokumenter/Om-oss/Jernbanestatistikk/Jernbanestatistikk-2008/`.

[15] Jernbaneverket (Norwegian National Rail Administration). *På skinner 2010*. Annual report. Oslo, 2010. URL: `http://www.jernbaneverket.no/no/dokumenter/2011/Multimedia/Pa-skinner-2010---Jernbaneverkets-arsmelding/`.

[16] Jernbaneverket (Norwegian National Rail Administration). *Punktlighet persontog pr mnd (%)*. [Online; accessed 2011-08-21]. URL: `http://www.jernbaneverket.no/PageFiles/14723/Punktlighet%20persontog%20pr%20mnd%20innev%C3%A6rende%20%C3%A5r.htm`.

[17] Jernbaneverket (Norwegian National Rail Administration). *Se punktlighetstall og tiltak*. [Online; accessed 2011-08-21]. URL: `http://www.jernbaneverket.no/no/Nyheter/Togenes-punktlighet-og-regularitet/`.

[18] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller and James W. Thatcher. The IBM research symposia series. New York: Plenum Press, 1972, pp. 85–103. URL: `http://www.cs.berkeley.edu/~luca/cs172/karp.pdf`.

[19] Victor Klee and George J. Minty. "How good is the simplex algorithm?" In: *Inequalities*. Vol. III. Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin. Academic Press, New York, 1972, pp. 159–175.

[20] Jan A. Koning. "Comparing the performance of ERTMS level 2 fixed block and ERTMS level 3 moving block signalling systems using simulation techniques". In: *Eighth International Conference on Computers in Railways. Computers in Railways VIII.* Ed. by John J. Allan et al. Southampton, UK: WIT Press, 2002. URL: `http://library.witpress.com/pages/dlfreepaper.asp?pID=89`.

[21] R. Duncan Luce and Albert D. Perry. "A method of matrix analysis of group structure". In: *Psychometrika* 14 (2 1949), pp. 95–116. ISSN: 0033-3123.

[22] Carlo Mannino and Alessandro Mascis. "Optimal Real-Time Traffic Control in Metro Stations". In: *Operations Research* 57.4 (2009), pp. 1026–1039.

[23] Ministry of Transport and Communications. *St.meld. nr. 16 (2008–2009) Nasjonal transportplan 2010–2019*. [Online; accessed 2011-09-24]. Oslo, Norway, 2009. URL: `http://www.regjeringen.no/nb?id=548837`.

[24] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. New York, USA: Wiley-Interscience, 1988. ISBN: 0-471-82819-X.

[25] Shmuel Onn. "Polynomial Time Primal Integer Programming via Graver Bases". In: *Wiley Encyclopedia of Operations Research and Management Science*. Ed. by James J. Cochran et al. John Wiley & Sons, Inc. [Published online], 2010. ISBN: 9780470400531. URL:

http://onlinelibrary.wiley.com/doi/10.1002/9780470400531.eorms0677/abstract.

[26] Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag Berlin Heidelberg, 2003.

[27] Joseph Y.-T. Leung Udaiprakash I. Gupta Der-Tsai Lee. "Efficient algorithms for interval graphs and circular-arc graphs". In: *Networks* 12.4 (1982), pp. 459–467. ISSN: 1097-0037.

[28] Robert J. Vanderbei. *Linear Programming, Foundations and Extensions*. Third edition. International Series in Operations Research & Management Science. Springer US, 2008. ISBN: 978-0-387-74388-2.