Strategies for the use of Data and Algorithmic Approaches in Railway Traffic Management.

Problem presented by

Ben Roullier

Resonate

ESGI130 was jointly hosted by
University of Warwick
Smith Institute for Industrial Mathematics and System Engineering

Report author

Dr Martine J. Barons (University of Warwick)

Executive Summary

A Railway Traffic Management problem can be defined as forecasting future progression of trains, identifying conflicts where two or more trains compete for available infrastructure, investigating options for resolution of conflicts, re-planning train schedules to minimise the impact on system performance. Performance management of complex networks is a problem common to a number of industries and applications. There has been much work over many decades on modelling the generation and optimisation of railway timetables. Much of this focuses on relatively simple railways and services and is therefore quite straightforward. Main line railways have a number of features that introduce significant complexity. Traditionally the problem of re-planning a timetable in near real time to manage and recover from service perturbations and disruption is simplified to help arrive at a solution in an acceptable amount of time, but this then can have unintended consequences which can amplify rather than reduce the disruption in the network. Resonate are interested in looking at different strategies / models / techniques for dealing with the problem, the likely strengths and risks of these, and how they might be adapted to improve existing solutions. The study group participants undertook a brief survey of recent literature on modelling train delays and found machine learning approaches, network models and a statistical approach to defining the efficiency of a station in dissipating delays which are worthy of further consideration. We then explored total of nine modelling approaches during the study group. The approaches fell broadly into two groups: those that sought to understand the propagation of delays (Approaches 1 to 6) and those that sought to offer strategies for minimising delays (Approaches 8 and 9). Approach 7 proposes a way of understanding the propagation of delays and using that to evaluate candidate policy decisions. There are a number of promising approaches here which provide useful lines of enquiry, many suitable for expansion beyond the simple railways modelled, to include variable train speeds, junctions and intersections, temporal differences in usage, such as tidal flows in and out of cities, and resource constraints.

Version 1.0
October 3, 2017
v+66 pages

Contributors

Martine J. Barons (University of Warwick)
Chris Bick (University of Oxford)
Marco Caselli (University of Warwick)
Antoine Choffrut (University of Warwick)
AC supported by European Research Council grant no. 616797
Vinh Doan (University of Warwick)
Payman Eslami (University of Warwick)
Jan Foniok (Manchester Metropolitan University)
Laura Guzmán-Rincón (University of Warwick)
Roger Hill (University of Warwick)
Emily Kawabata (University of Edinburgh)
Giovanni Mizzi (University of Warwick)
Chris Norman (University of Warwick)
Norbert Peyerimhoff (Durham University)
Karina Piwarska (University of Warsaw)
Colin Please (University of Oxford)
Caoimhe Rooney (University of Oxford)
Sofia Trejo (Imperial College London)
Luke Whincop (University of Warwick)
Robert Whittaker (University of Warwick)
Jessica Williams (University of Oxford)
Yuanwei Xu(Imperial College London)

Contents

1 Introduction **1**

2 Problem statement **2**

3 The solution **3**
3.1 Approaches in Literature . . . 3
3.2 Approach 1: Toy model . . . 6
3.2.1 Results . . . 10
3.2.2 Discussion . . . 10
3.3 Approach 2: Signalling Dynamics . . . 13
3.3.1 Signalling Background . . . 13
3.3.2 The model . . . 14
3.3.3 Initial model and numerical implementation . . . 15
3.3.4 Results and discussion . . . 16
3.3.5 Further work (model refinement) . . . 17
3.4 Approach 3: Continuum Theory and Flux Modelling . . . 18
3.4.1 Classic Traffic Modelling . . . 18
3.4.2 Modified Velocity Function . . . 18
3.4.3 Effect of Signals . . . 20
3.4.4 Conclusions . . . 21
3.4.5 Flux Modelling . . . 21
3.4.6 Conclusion . . . 24
3.5 Approach 4: Predictive Modelling . . . 24
3.5.1 Method A: Data driven method . . . 24
3.5.2 Method B: Event based delay propagation in a train network 27
3.6 Approach 5: Identifying the biggest contributors of delay in the network 27
3.7 Approach 6: A Bayesian model of route status . . . 28
3.7.1 Problem to address . . . 28
3.7.2 Description of model . . . 28
3.7.3 Further development . . . 30
3.8 Approach 7: Bayesian Networks and IDSS . . . 32
3.8.1 Method . . . 32
3.8.2 Results . . . 32
3.8.3 Discussion . . . 35
3.8.4 Future developments . . . 41
3.9 Approach 8: Mixed Integer Programming . . . 41
3.9.1 Decision variables . . . 41
3.10 Approach 9: How to improve global outcome from local decisions? . 43

4 Conclusions **44**

A Appendices **45**
A.1 First appendix: Toy Model code in Python 2 . . . 45
A.2 Second Appendix: Perl code for Signalling Dynamics . . . 55

A.3 Third appendix: Code in *Mosel* modelling language to be solved by *Xpress* for Mixed Integer Programming 57
A.4 Fourth appendix: Code in *R* for A Bayesian model of route status . 61

References **64**

1 Introduction

(1.1) There has been much work over many decades on modelling the generation and optimisation of railway timetables. Much of this focuses on relatively simple railways and services and is therefore quite straightforward. Main line railways have a number of features that introduce significant complexity:

- Routes with many junctions and intersections
- Variable train speeds
- Variable train stopping patterns
- Mixed passenger and freight traffic
- Peak patterns / off peak patterns
- Tidal flows in and out of cities for the working day
- Constraints from resources (trains and crew)

As the volume of traffic in the system increases, the effect of perturbations in the actual progress of trains can introduce unstable behaviour that provides significant challenge for the compromise between performance and capacity. Traffic management systems are therefore needed to forecast the likely future progress of trains, identify conflicts, and modify the planned schedule of trains to minimise the resulting disruption and accelerate recovery back to the planned service.

Resonate is a technology company specialising in rail and connected transport solutions. We have a powerful platform and an excellent team that is helping us to support numerous elements of the Digital Railway initiative being driven by Network Rail and the DfT. We are also working hard to help deliver intelligent traffic management and smarter cities internationally. We have over 50 years of rail industry experience and used to be the research division of British Rail before being privatised in 1996. Our understanding spans safety critical signalling control, rail operations management, logistics and IT. We have combined our rail background knowledge with agile development methods, data gathering, advanced algorithms and the latest cloud computing, so that we have the tools to deliver 21st century traffic management.

In 2016 we changed our name from DeltaRail to Resonate in recognition of the fact that we are entering a new and demanding age of connected and intelligent transport. We have embarked on a drive to maximise capacity and performance through predictive intelligence, shared data, joined up travel and informed customer journeys.

2 Problem statement

(2.1) A Railway Traffic Management problem can be defined as:

- Forecasting future progression of trains
- Identifying conflicts where two or more trains compete for available infrastructure
- Investigating options for resolution of conflicts
- Re-planning train schedules to minimise the impact on system performance

Traditionally the problem of re-planning a timetable in near real time to manage and recover from service perturbations and disruption is simplified to help arrive at a solution in an acceptable amount of time, but this then can have unintended consequences which can amplify rather than reduce the disruption. We would be interested to look at different strategies / models / techniques for dealing with the problem, the likely strengths and risks of these, and how they might be adapted to improve existing solutions. Performance management of complex networks is a problem common to a number of industries and applications and therefore it is likely that approaches already exist that could be adapted for use in rail. Detailed data on planned and actual services over many months is available to support the workshop.

Constraints

A number of constraints can complicate the timetable optimisation and have a significant impact on the robustness of a solution:

- Availability of infrastructure – typically during disruption, the level of infrastructure available is reduced, either by a failure of the equipment, or a blockage caused by train failures or human intervention.
- Availability of rolling stock (their location, and the fact that there are various different types of train, with differing requirements for maintenance and capacity)
- Availability of staff (location, ability to drive certain trains & routes, shift hours).

Costs & Benefits

The primary benefit of improved railway performance is of course to the passengers, freight operators and the UK economy. In addition to this:

- The Rail Regulator can fine Network Rail for failure to meet performance targets – the last fine was £53m in July 2014.
- There is a delay attribution regime in place between Network Rail and Train Operators where the originators of delay pay compensation to those who suffer delay – typically £100m can change hands across the industry in a year.

- There are additional costs for reimbursing customers for delays or reduced services.
- Customer service is important, often measured indirectly through the number of trains calling at each station compared to planned operation.

There is therefore also a good business case for minimising train delays

3 The solution

(3.0.2) We undertook a brief survey of recent literature on modelling train delays and found machine learning approaches, network models and a statistical approach to defining the efficiency of a station with respect to delays. We explored total of nine distinct modelling approaches during the study group and The approaches fell broadly into two groups: those that sought to understand the propagation of delays (Approaches 1 to 6) and those that sought to offer strategies for minimising delays (Approaches 8 and 9). Approach 7 proposes a way of understanding the propagation of delays and using that to evaluate candidate policy decisions.

3.1 Approaches in Literature

(3.1.1) **Literature** *Primary delays* are caused by unexpected stochastic events in the system (e.g. technical faults, prolonged alighting/boarding times, adverse weather conditions etc.); these can have a knock-on effect to create *secondary delays* on other services. The propagation of primary delays depends significantly on railway timetabling and infrastructure and they can have an impact on services both spatially and temporally far from the origin. This makes the prediction of secondary delay challenging, but numerous approaches have been considered in recent literature. An overview of recovery models and algorithms for real-time railway rescheduling can be found in [9]. Rescheduling models for railway traffic management in large-scale networks can be found in [10].These papers above contain a great many references to other relevant literature.

(3.1.2) **Modelling train delays** with *q-exponential* functions is used in [11] to provide an efficiency score for each station. *q-exponentials* are defined as $e_q(x) = (1+(1-q)x)^{1/(1-q)}$, where q is a real parameter, the entropic index. These model complex systems with fat-tailed distributions. Briggs et al. collected most recent delay data on over two million train departures at 23 major stations between September 2005 and October 2006, including over 200,000 departures for Manchester Piccadilly, one of the busy stations. Using the model $e_{q,b,c}(t) = c(1-b(q-1)t)^{1/(1-q)}$, they estimated the parameters using nonlinear least squares. Then q measures the deviation from

an exponential distribution , so an estimated q larger than unity indicates a long-tailed distribution. 80% of the trains recorded $t = 0$ indicating a delay of less than 1 minute, so this model represents the conditional probability of delay given the train is delayed 1 minute or more. Assuming the waiting time distribution is given by a Poisson process $P(t|\beta) = \beta e^{-\beta t}$, allowing β to fluctuate to describe the temporal variations in the rail network due to weather, holidays, signal failures etc., the degrees of freedom in the model. When β is small, delays are more frequent. From the model, the average contribution of each degree of freedom is estimated form the fitted value of b, giving a statistic $\langle \chi_i^2 \rangle = \frac{1}{2}(q-1)b$ which is laarge when a local station is doing well, i.e. the local exponential decay of the delay times is as fast as it can be. Stations with the same q (external degrees of freedom in the network) can usefully be compared. This analysis showed that Cambridge ans Edinburgh were the best performing busy stations under this criterion.

(3.1.3) **Data Mining** Recent advances in data mining and machine learning techniques have enabled the efficient analysis of large data sets for accurate prediction. Several instances of applications of these techniques to train delay can be found in literature. [12] use support vector regression to identify the relationship between various system characteristics and train delay; [14]use artificial neural networks to predict delay, achieving high accuracy in an application to Iranian railways. A major flaw in these approaches for our application can be the computational time required for the analysis of very large data sets; [13] propose a fast learning algorithm based on the 'Extreme Learning Machine', which can extract relevant information quickly to make accurate predictions about future network states, they show the method can improve the current prediction systems implemented in Italian railway networks.

(3.1.4) **Petri Nets** Petri nets (after Carl Adam Petri) are modifications of simple network models, often used in the modelling of discrete distributed systems and in particular, more recently, railway networks. A Petri net is a bipartite graph, with directed edges between two sets of nodes: transitions (representing events like arrivals or departures of trains from track stretches or stations) and places (representing activities like travelling from one track stretch to another or conditions like waiting for passengers to change trains). The dynamics is modelled by the movements of tokens through the places, representing events like arrivals and departures of trains. Tokens travel through the Petri net via enabled transitions. Trains can be represented by (coloured = distinguishable) tokens, other types of tokens can be used to model capacity restrictions (for example maximal number of tracks in a station) or to realise train interconnections. A very understandable introduction into Petri nets and their modelling of train service intention and scheduling can be found in Chapters 5.1-5.3 of [15]. This PhD thesis introduces a measure of capacity of station regions as well as

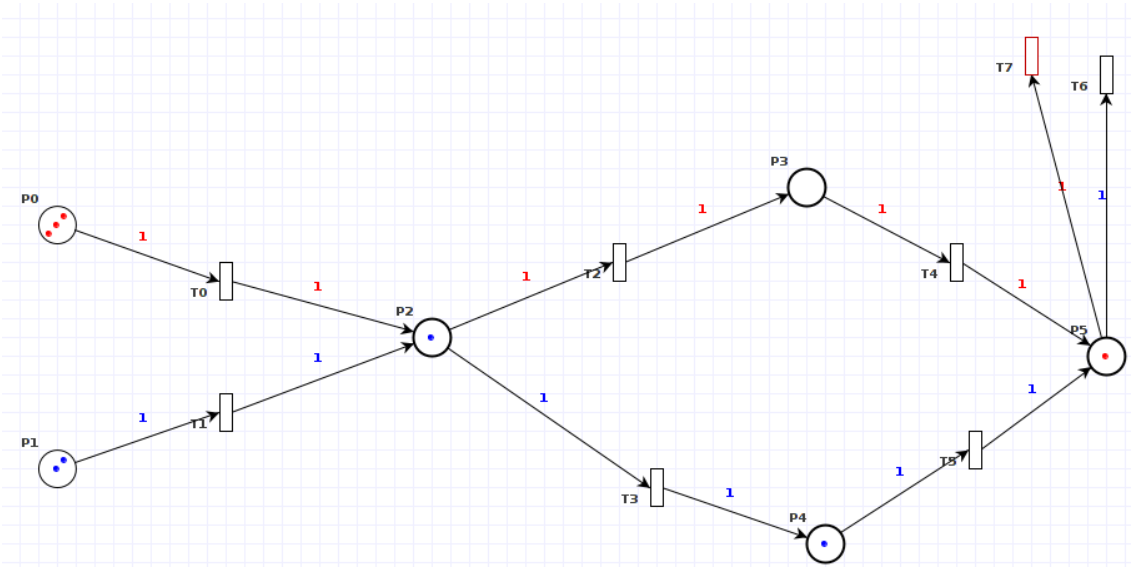


Figure 1: A simple High-level Petri nets (HLPNs) representation created using PIPE. Circular and rectangular nodes represent places and transitions respectively. In this model, two types of train are included, 'fast' red tokens, and 'slow' blue tokens. This image illustrates implementation in one of several software packages available, and widely used, for these problems.

for stability of a timetable and is openly accessible at

`https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/46806/eth-28137-02.pdf`

(3.1.1) When additional values and parameters are included, specification of multiple token characteristics for example, such structures are referred to as high-level Petri nets (HLPNs), see Figure 1. A simple HLPN created using PIPE. Circular and rectangular nodes represent places and transitions respectively. In this model, two types of train are included, 'fast' red tokens, and 'slow' blue tokens. This image illustrates implementation in one of several software packages available, and widely used, for these problems:

`https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`

There is a substantial amount of literature of railway simulation using petri nets. Here is a selection of papers concerned with this topic. Articles dealing with safety aspects: [27], [16]. An article dealing with conflict forecasting and delay minimisation: [17].Articles dealing with train scheduling and its optimisation: [18], [19], [20], [21]. Several articles have focussed on 'on-the-fly' responses to a primary delay, a small selection of which are provided here: [25], Inputs to a discrete Petri net model are 'fuzzified' to create a probabilistic prediction of delay propagation; Success is shown from testing the model on part of the Belgrade railway node. [22], [23] similarly model delay propagation probabilistically by considering stochastic Petri nets., [24], [26].

(3.1.1) HLPNs have been used as models of railway networks for multiple purpo-

ses, a large body of the literature focuses on optimisation of timetabling, which occurs several months ahead of dispatch. Several studies however have focused on 'on-the-fly' responses to a primary delay, a small selection are provided here. Milinkovic et al. (2013) 'fuzzify' the inputs to a discrete Petri net model to create a probabilistic prediction of delay propagation. They show success from testing the model on part of the Belgrade railway node. Caetano and Teixeira (2014) similarly model delay propagation probabilistically by considering stochastic Petri nets. [24] build a prototype tool for identification of conflicts and estimation of knock-on delay, tested, with success, on the Dutch railway network.

3.2 Approach 1: Toy model

(3.2.1) The idea is to create a very simple model in order to simulate a train line at the tactical level. We build a exclusive interacting particle system on a network which represents the line.

(3.2.2) The network is made of three fundamental components:
Straight: this is a pieces of track included between two signals. For the purpose of the simulation, the time step is the time needed for a train to move from one berth to the following, and it set at 2 minutes. The time a train stops at a station is also set at 2 minutes. Trains may only proceed to the next node if the next node is not occupied by another train.
Split: this is a berth followed by a signal that involves a decision to send the train to one of n possible following berths. In the network we consider here, $n = 2$ always; In reality, this is realistic because points/switches on the railway can only take two positions. More complex junctions are achieved by having sets of points in series.
Join: this is a berth preceded by a signal that involves the decision is to allow one of the trains on the previous $n = 2$ berths to proceed to the next one and stop all the others.

(3.2.3) Using these simple components other structures can be built :
Station: this is a combination of a split, $n = 2$ straights, and a join;
Overtake track: this is similar to a station, but between the split and the join, the two tracks have a different number of straights. We are not considering this structure in our simple model since overtaking can also happen in a station.Figure 2 shows all the fundamental components of the network and the sample train network that we will use for our simulations.

(3.2.4) *Important remarks:* For coding purposes, in the network above berths are represented by nodes. Signals are at the end of nodes, as they would in a a real berth. We consider two different kinds of trains:
Fast trains: scheduled to go from (1) or (2) to (17) without stopping at

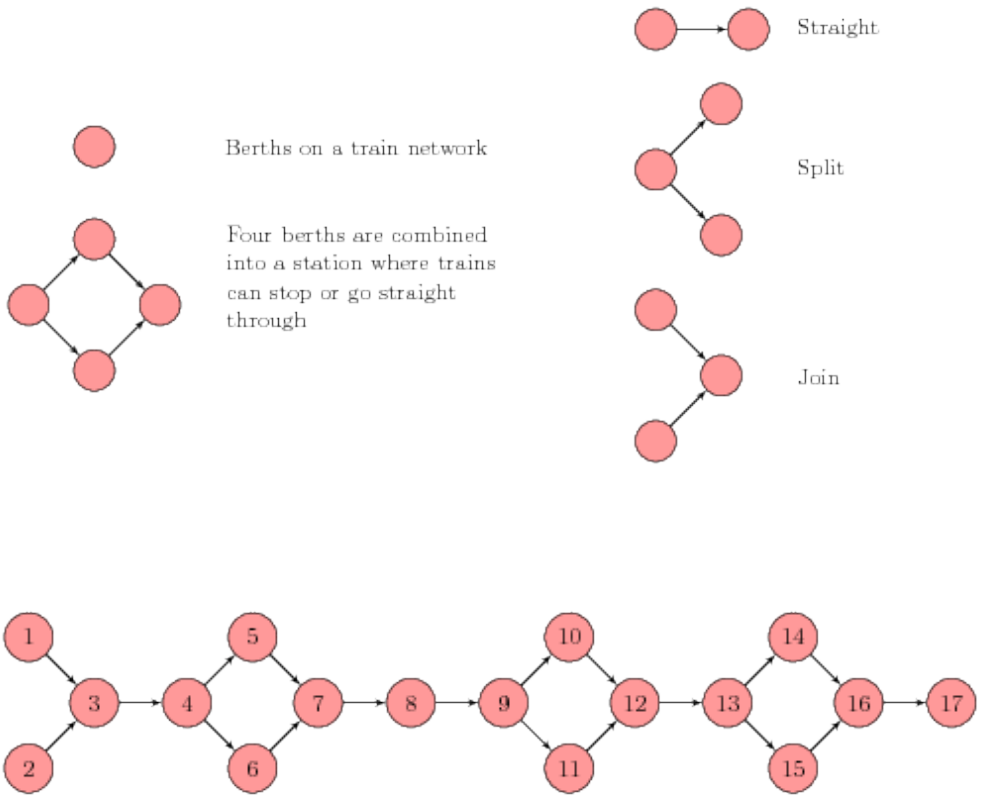


Figure 1: Simple train network representation

Figure 2: Fundamental components of the network and the sample train network used for simulations.

the stations;
Slow trains: scheduled to stop at every station.
In our model, stations correspond to the sets of berths (5,6), (10,11), and (14,16). In our model, both types of trains move at the same speed and fast trains have priority at splits and joins; this reflect the fact that, often, the speed of a train is constrained by the maximum speed permissible on the track, rather than by its intrinsic speed limit. Moreover we assume that any berth has the same length. Other kinds of trains could be added to the model later. For example, one could consider trains which only stop at the station (10,11).

(3.2.5) **How does the model work?** Trains are injected in nodes (1) or (2). For now the frequency is fixed, depending of how much we want to saturate the network. We assume that trains in node (17) always disappear. The directed network is given as an adjacency matrix. Then it is necessary to assign transition probabilities to each berth, i.e. applying a transition matrix to the whole system. The transition matrix is different at every time step and needs to be recalculated in a particular order, starting from the terminal node, i.e. when we know what the train in node (17) does, we decide what the node in (16) can do; after we know that we can decide what will happen to trains in (14) and (15) and so on.

(3.2.6) Join nodes require particular attention. To decide the strategy at these points all nodes preceding it must be processed together. In general, the decision is made by the following algorithm:

```
‘‘‘pyton
if ’the join berth is occupied’ then
    ’all the trains immediately before will receive the signal to stop’
    else
    if ’there are more than one train in the station’ then
        if ’there is a fast train’ then
            if ’there are multiple fast trains’ then
                ’the one with the higher delay goes first’
                 # in this case higher time since injection
                else
                ’the fast train goes first’
            else
            ’the train with the highest delay that has already stopped
goes first’
        end
        else
        ’the train can proceed’
    end
‘‘‘
```

How does this address the problem? When we simulate the system filling it below a certain capacity level, trains are not expected to interfere with one another. In this case it is trivial to calculate the time it would take to a train to complete the entire journey. On the other hand, by running the simulation many times at different capacities we can record the distribution of the time taken for trains to complete the whole journey. Taking the mean time or other suitable statistic, we can produce a timetable. Once a time table has been established, we could then study how delays can propagate in the network and answer questions such as what happens when we are near full capacity? Which delays considerably affect the performance of the network? What happens if we introduce injection delays and ejection delays? What happens if we make a train stop at one station for an unknown amount of time? We can change the join and split rules in to explore different scenarios and study how these affect the network. For example, knowing that a fast train is delayed could trigger the action of making slow trains wait in previous stations. This could be relevant when considering trains having different speeds.

Results to date Our network is programmed to favour fast and delayed trains. To do so we set the following rules for a train to move forward:

- Fast trains always have priority over slow trains.
- If both trains have the same type and the same amount of delay (which could be zero) the probability of moving forward is 50%.
- If both trains have the same type and one of them is more delayed than the other one, the most delayed train moves first.
- In stations one platform has preference. The second platform only gets used if the first one is taken.

The current model is as simple as possible. This means that splits and joints are related to three berths, and stations have two platforms. In future development, adjustments could be made to model a network which is a more accurate representation of an existing railway network.

Code The model simplify the rail network, representing it as a finite graph where each node corresponds to a berth, consistently with the information available at the global level. At each time step the program, in function of the current global status of the system, decides (and applies) the next move of all the trains in the network. The main function is `'railtrack'`, it has basically one input and several settings. The input `'ad_matrix'` is the matrix which describes the network connections as a graph. The parameter `'time_steps'` determines the number of iterations. Other parameters refer to delays probabilities and congestion:

```
prob_inject = floating number between 0 and 1, probability that in a
single starting empty berth is filled with a train
lambda_delay = non negative value, constant for the poisson
describing the delay of the trains, the constant corresponds to the
```

```
average delay
prob_stop_anywhere = floating number between 0 and 1, probability
that a train spends one time step further in the station
prob_stop_station = floating number between 0 and 1, probability
that a train spends one time step further in any non- station berth

The outputs are the:
trains_old  = set containing all the trains that arrived at the
last station.
trains_now  = list of all the berth at the end of the computation,
 the entry is the class of the train in the berth, 0 if there are
no trains in that edge.
transitions = matrix with the instruction for moving the trains at
 each time step. Ex: transitions[7] it is a vector that describes
 the action to be taken at the time_step=7, the train in the
position 4 goes in the position transition[7][4], if in a specific
position there are no train the relative vector entry is null.
matrix_of_trains = matrix with the train in the network at each
time step (characterized just by the category), used for printing a gif output.
```

Moreover there are some utility functions, in particular `'outputcsv'` generates CSV files useful for studying the behaviour of the network with different parameters. See Appendix A for full code.

3.2.1 Results

(3.2.7) We ran the simulation with different settings of parameters (more than 1000 different settings). Here it is how our network looks (blue are fast trains, red slow trains):

(3.2.8) Figure 3 is an example of the kind of analysis you can do with the model. Blue dots are fast trains, red slow. These show the impact of the strategy on the initial delay: if a train has more delay then it will not gain more delay during the journey since it has acquired priority. A better analysis would describe the propagation of the delay over the network. Figure 4, refers to the following settings: injection probability = 0.75 (probability that a train enter in a start edge); lambda Poisson coefficient = 5 (distribution on the initial delays of the trains); probability of stopping anywhere 0.001; probability of stopping at station 0.01.

3.2.2 Discussion

(3.2.9) **Strengths** The code has the potential of modeling the whole rail network. It has different setting that can simulate peak times, tendency of delays in stations and stops at any berth. One could use this code to study the

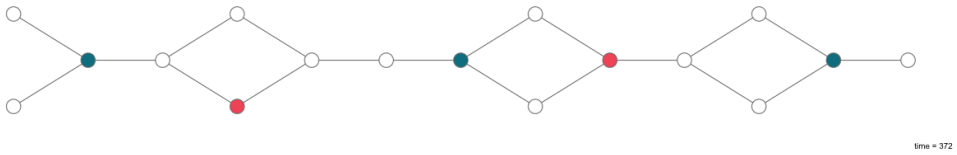


Figure 3: An example of the kind of analysis you can do with the model. Blue dots are fast trains, red slow. This shows the impact of the strategy on the initial delay: if a train has more delay then it will not gain more delay during the journey since it has acquired priority

propagation of delays and test the implementation of new decision making schemes. When a 'critical' decision has to be taken it would possible to run several forecasts of the network with different decisions and pick the method which most probably will minimize the delays.

(3.2.10) **Limitations** The actual code is just a sketch written in few days, therefore it is flexible to a certain extent. Although it is possible to input a rail network by its adjacency matrix, there are some constraints about the structure of it. For example, the current code does not deal with overtaking (they are just possible in the stations), nor with paths of different lengths. Time, distance and velocity are treated as discrete values. So far any train takes one time step to move to the next berth, it would be possible to describe the network with a more flexible and realistic structure (including trains with different speeds and subdividing actual "long" berth in shorter ones) but we retain that the discrete approach is a key point in order to make the simulation feasible from a computational point of view.

(3.2.11) **Improvements** Certain generalizations to the code can be made quite easily, other ones would require deeper modifications. For further imple-

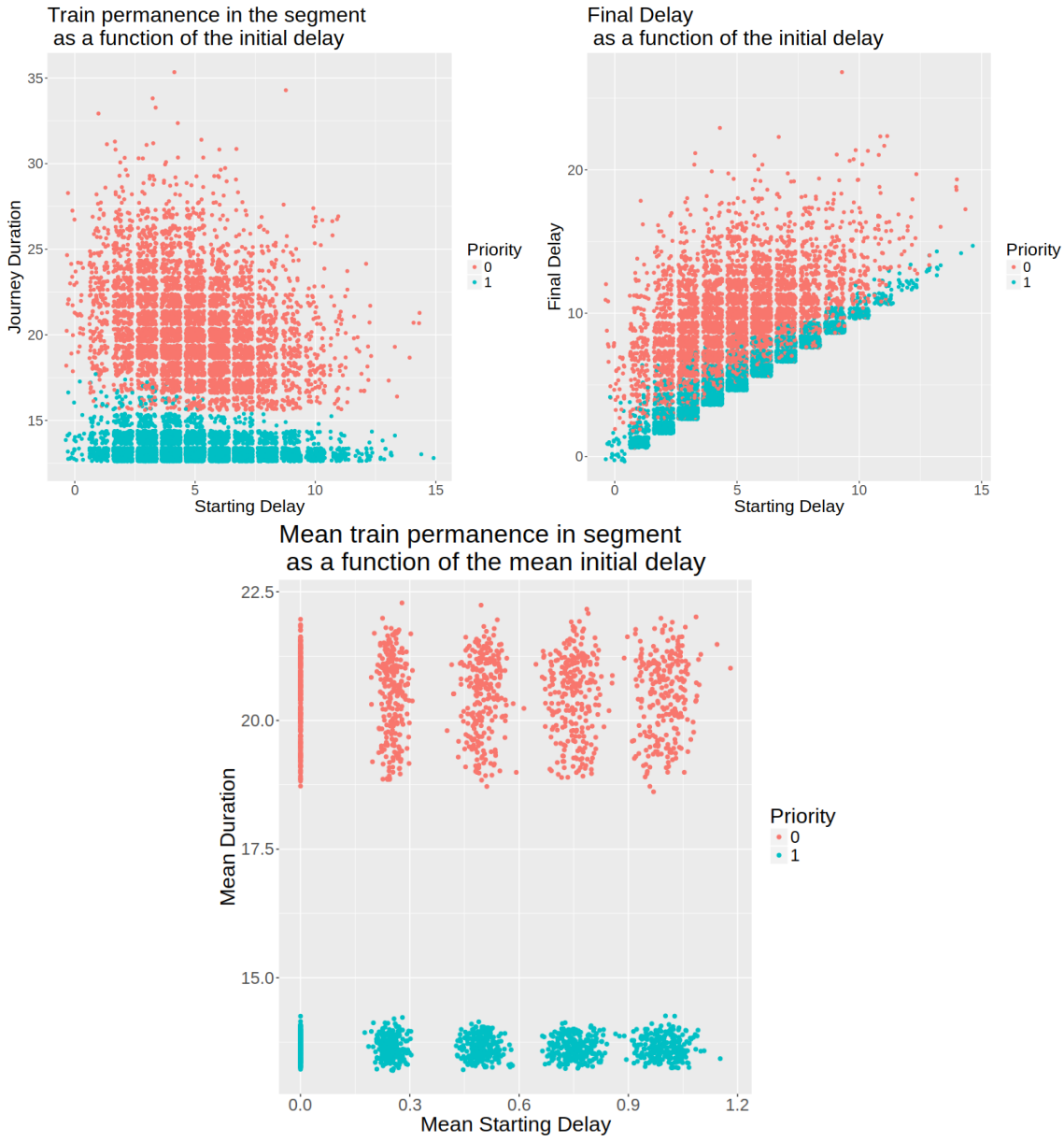


Figure 4: injection probability = 0.75 (probability that a train enter in a start edge) lambda poisson coefficient = 5 (distribution on the initial delays of the trains) probability of stopping anywhere 0.001 probability of stopping at station 0.01

mentation it would be key to rewrite the code in an a faster language, such as C or Fortran. This would allow to process the simulations as fast as possible and would give the possibility of obtaining sensible data in real time. One possible approach would be forecasting the network behaviour accordingly to different strategies (running several simulation in few seconds) and pick the best one.

(3.2.12) It would be possible to model the parameters in such a way that the simulation is able to address realistic situations, for instance by using

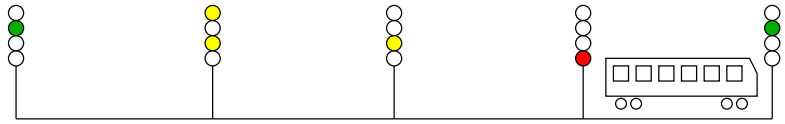


Figure 5: A sketch showing the signal aspects in the blocks behind a train. The first block behind shows red, the next one yellow, the one after that double-yellow, and finally green.

mixed strategies for the choices depending on the current status of the whole system, of the following track or any particular conditions that might be interesting to explore.

(3.2.13) Since as far as we understand there is not yet a clear algorithm, or a standardized way to take decisions, this approach could help developing a global decision making rule set by observing how every different kind of node most commonly behaves.

(3.2.14) **Contacts** Marco Caselli
Laura Guzmán-Rincón
Roger Hill
Giovanni Mizzi
Sofia Trejo

3.3 Approach 2: Signalling Dynamics

3.3.1 Signalling Background

(3.3.1) We assume that line-side signals are placed at roughly regular intervals along each line of track, and are the primary (and often the only) mechanism for controllers to affect the progress of a train along its route. Signals show one of four aspects: green, double-yellow, yellow and red. At a green signal the driver can proceed at maximum speed for the track, at double-yellow and yellow the train must slow by increasing amounts and be prepared to stop at a future signal. A red signal must not be passed.

(3.3.2) The track is divided into blocks by the signal locations. A built-in safety feature ensures that if there is a train in next block then the preceding signal shows red. If the next block is free but the one after contains a train, then the signal is yellow, and if there are two free blocks before the next train, then the signal is double-yellow. With three or more free blocks the signal shows green. See figure 5

(3.3.3) These signals can be manually over-ridden, but only to keep a signal at an aspect that is *less* permissive than the safety settings above. It is generally

not permitted to manually change a signal to a *more* permissive aspect, since this might result in trains not having enough warning to stop at a future red. Thus by default, signals are left on red until a route is allocated to a train wishing to travel along a particular stretch of track.

3.3.2 The model

(3.3.4) We used discrete train model to describe the local behaviour of individual trains as they encountered signals on the track and the feedback between multiple successive trains on the same line. In particular we are interested in the effects of perturbations on a string of trains running under green lights, and on the effects of a string of trains either coming to a halt at an unexpected red, and on a string of stopped trains starting up again after a red light turns green. We take into account the finite length of each train, the fact that signals can be seen in advance of their position, and the time is takes to accelerate and decelerate a train to its desired speed.

(3.3.5) We consider a set of N trains running on a single track with K equal signal blocks of unit length. We index the trains by $n \in \mathbb{Z} \cap [1 : N]$ and the signals by $k \in \mathbb{Z} \cap [0 : K]$. We denote time by t and distance along the track by x.
The dependent variables in the problem are then

(3.3.6)

$x_n(t)$ — the position of nth train at time t
$v_n(t)$ — the velocity of nth train at time t
$s_k(t)$ — the aspect of the kth signal at time t
$\sigma_n(t)$ — the aspect of last signal that train n has seen as of time t

(3.3.7) where the signal aspects are encoded as 0 for red, 1 for yellow, 2 for double-yellow, and 3 for green.
The input functions and parameters of the model are as follows:

(3.3.8)

h — the distance ahead that a train driver can first view each signal
d — the length of each train
$\mathcal{A}(v, \sigma, \delta)$ — the acceleration function

where the acceleration function can depend on the current speed v, the last observed signal aspect σ, and distance to next signal δ.

(3.3.9) The four governing equations are then as follows. The rate of change of position of each train is given by its velocity, and the rate of change of velocity is given by the specified acceleration:

(3.3.10)

$$\frac{\mathrm{d}x_n}{\mathrm{d}t} = v_n \tag{1}$$

$$\frac{\mathrm{d}v_n}{\mathrm{d}t} = \mathcal{A}(v_n, \sigma_n, \lceil x_n \rceil - x_n) \tag{2}$$

(3.3.11) Unless overridden by a manual intervention, the signal aspects are determined by the number of blocks ahead of each signal the closest train is. We must consider both the front and back of each train, and so obtain

(3.3.12)

$$s_\alpha = \min\Big\{ \min_{n;\, x_n > \alpha + d} (\lfloor x_n - d \rfloor - \alpha), \min_{n;\, x_n > \alpha} (\lfloor x_n \rfloor - \alpha), 3 \Big\} \tag{3}$$

(3.3.13) Finally, the signal last seen by each train is determined cases, depending on whether the train is in sight of a signal. If it is, then we adopt that aspect. If not, then we leave the last observed aspect unchanged. Hence:

(3.3.14)

$$\begin{cases} \sigma_n = s_{\lceil x_n \rceil} & : \quad \lceil x_n \rceil - x_n < h \\ \dfrac{\mathrm{d}\sigma_n}{\mathrm{d}t} = 0 & : \quad \text{otherwise} \end{cases} \tag{4}$$

3.3.3 Initial model and numerical implementation

(3.3.15) For an initial model, we take $h = 0.3$ and $d = 0.1$. For the acceleration function, we assume there is a desired speed $\mathcal{V}(\sigma)$ for each signal aspect, and then take a simple smoothed constant-acceleration function to drive the speed towards the desired one:

(3.3.16)

$$\mathcal{A}(v, \sigma, \delta) = a_0 \tanh((u - \mathcal{V}(\sigma))/v_0) \tag{5}$$

(3.3.17) For the target velocities, we take

(3.3.18)

$$\mathcal{V}(0) = 0, \qquad \mathcal{V}(1) = 1, \qquad \mathcal{V}(2) = 1.8, \qquad \mathcal{V}(3) = 2. \tag{6}$$

(3.3.19) The acceleration parameters are set as $a_0 = 3$ and $v_0 = 0.1$.

(3.3.20) The model is implemented numerically using the Perl code in Appendix A.2. Each of the variables x_n, v_n, σ_n, and s_α are updated in turn, based on the equations above, with simple forwards-Euler time-stepping for the temporal derivatives.

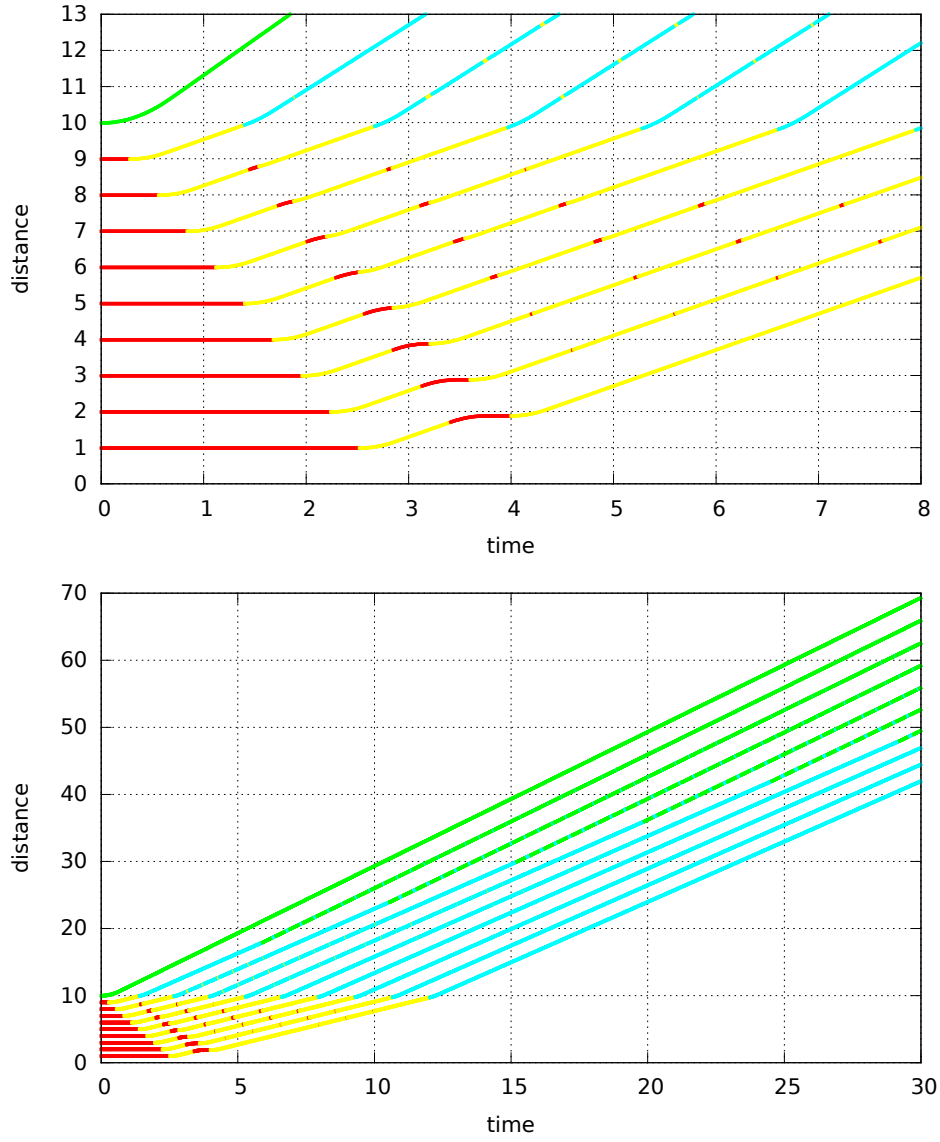


Figure 6: Distance–time graphs showing a set of trains restarting after being held at a red signal. Each line represents a train, and the colour is the aspect of the signal (cyan represents double-yellow). The velocity model and parameter values are as specified in §3.3.3.

3.3.4 Results and discussion

(3.3.21) To illustrate the model, a simple case is studied, simulating trains starting up from a line blockage that has just been cleared. A line of trains is initially at rest, with one train per block each stopped just behind a red signal. The signals ahead are then turned green allowing the first train to set off. As each train clears its block the train behind can then start to

move.

(3.3.22) The results of this model (with the parameter values above) are shown in figure 6. We observe and interesting instability in the start-up process, whereby trains further back initially start moving, but then have to stop again at a red light.

(3.3.23) When trains start up in order, the following train always loses a bit of ground on the train ahead as it only starts to accelerate once the signal ahead turns to yellow. So once up to speed it would be running slightly more than a block length behind. The instability is caused by the driver being able to see the next red signal further ahead than this lost ground. Thus a red signal is observed in the next block immediately after restarting. Whether or not this causes a train to stop, it will likely make it at least slow, reducing the distance to the train behind. Hence the next train in the line will be affected slightly more, and so on.

(3.3.24) The instability only grows on the first new red, as the additional slowing for that red allows the train ahead to make up enough ground to be further ahead by the next signal.The only way to avoid this instability is to ensure that the distance lost during acceleration is large enough that each train has no need to break for the next signal. This could be achieved by having trains accelerate more slowly, and/or introducing a more realistic breaking model that doesn't react immediately to an observed red signal, but only starts to slow the train when δ is small enough that braking is needed to come to a halt before the signal.

(3.3.25) We also observe that the trains spend a long time running under double-yellows, before being far enough apart to only observe green lights. This is because of the relatively small speed differential in the model between green and double-yellow. Thus when a train in front is continually passing green lights and the next train is just over two blocks behind and so seeing double-yellow, the train ahead is only going slightly faster than the train behind. It therefore takes a long time for the train ahead to make up enough extra ground to extend the gap behind it to three blocks.

3.3.5 Further work (model refinement)

(3.3.26) This is only a preliminary model, but is designed to show how detailed train-level modelling could be accomplished. A number of refinements would need to be made for this model to have practical applications.

(3.3.27) These include:

- Allowing for variable block lengths and signal observation distances;
- Allowing for different trains to have different characteristics;

- Providing more realistic acceleration and deceleration models.

In particular, the function $\mathcal{A}(v, \sigma, \delta)$ should be adjusted so that breaking only commences when necessary to adjust the speed to the desired value as the signal is passed. So for δ larger than some critical value (dependent on v and σ) no breaking would occur. Currently breaking commences the moment a new signal aspect is observed.

Contact: Robert Whittaker

3.4 Approach 3: Continuum Theory and Flux Modelling

3.4.1 Classic Traffic Modelling

(3.4.1) Modelling the flow of traffic to understand the formation and evolution of traffic jams is a well-studied problem, and the ideas can be extended to consider traffic jams on trains. The governing equation, imposing 'conservation of vehicles' is

(3.4.2)
$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0, \tag{7}$$

(3.4.3) where $\rho(x, t)$ is the number density of vehicles, $u(x, t)$ is the speed, and thus the flux is given by ρu. Independent variables x and t are distance along the track and time, respectively. Classically the speed is assumed to be a simple function of density with effects such as acceleration and driver response times neglected. In this case the conservation equation above can be formulated as a differential equation for ρ. Most commonly for car traffic, the speed-density relation is taken to be

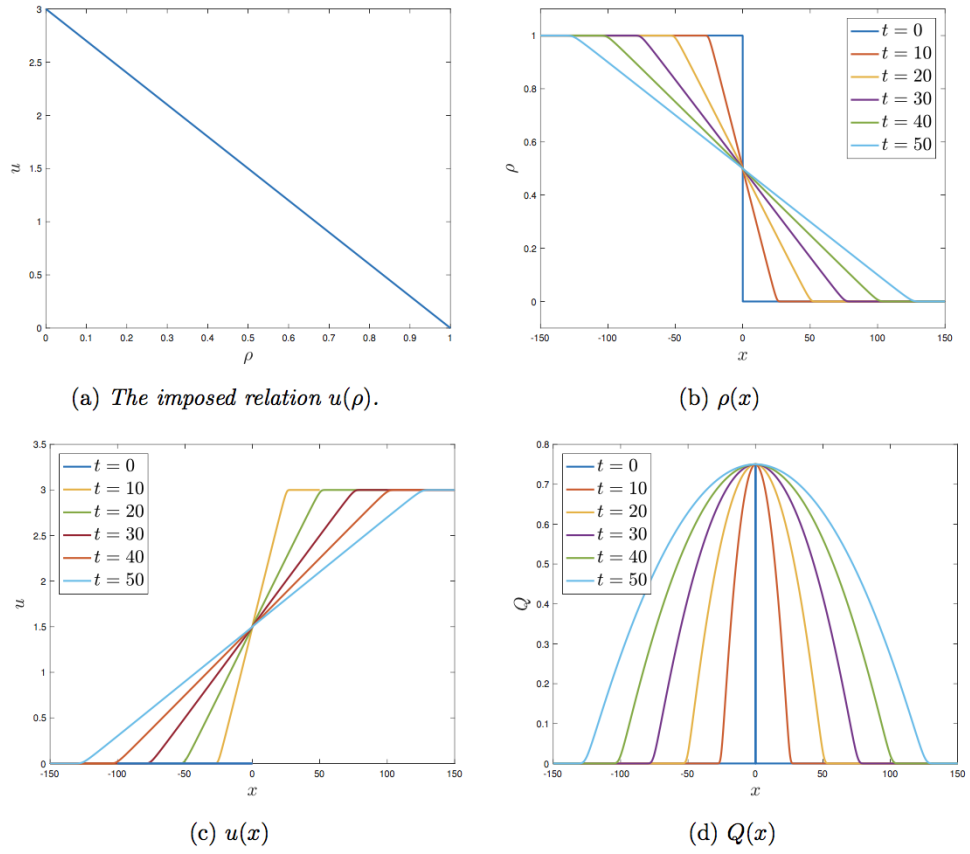
(3.4.4)
$$u(\rho) = 1 - \frac{\rho}{\rho_{\max}}, \tag{8}$$

(3.4.5) where $\rho_{\max}$ is the maximum density of vehicles (corresponding to a standstill traffic jam) and the maximum speed is $u = 1$.

(3.4.6) Using this model we can consider a simple situation in which the initial density of vehicles is zero for x positive and ρ_{max} for x negative, representing a line of vehicles stuck at a red light. The resulting time evolution of the solution to the conservation equation with the velocity function given by the above equation is shown in Figure 7.

3.4.2 Modified Velocity Function

(3.4.7) To adapt the classical theory to describe the flow of train traffic, as opposed



(a) *The imposed relation* $u(\rho)$. (b) $\rho(x)$

(c) $u(x)$ (d) $Q(x)$

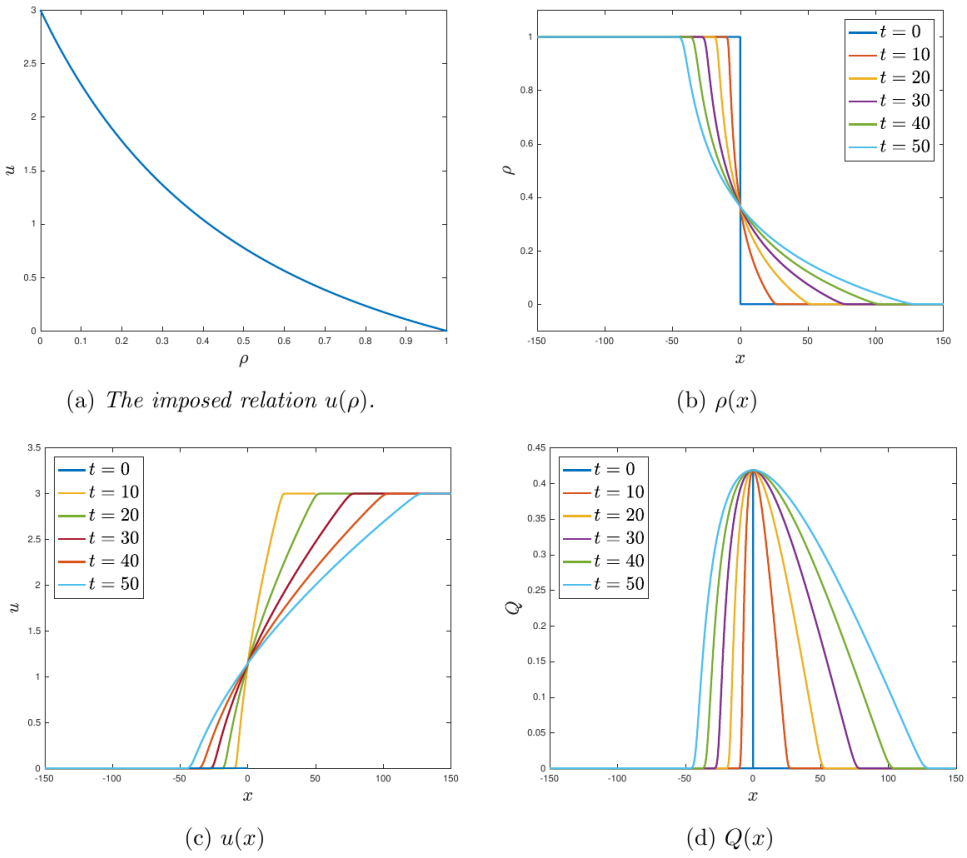
Figure 1: *Evolution of the solution to the conservation equation, with* $u(\rho)$ *given by the equation above, at various time points. Equation solved numerically using Godunov's scheme with an initial distribution of* $\rho(x) = 1 - H(x)$. *Spatial domain is* $-150 \leq x \leq 150$, *and boundary conditions are* $Q(-150) = 0$, $\partial Q/\partial x(150) = 0$.

Figure 7:

to cars, we discussed a number of options. One of these was to take the formula

$$u(\rho) = u_{\max}\left(\frac{4}{3(1+\rho/\rho_{\max})^2} - \frac{1}{3}\right), \tag{9}$$

where the non-linearity of this relation is based on the concept that the driver attempts to choose a speed that will allow them to stop in a given distance should the signalling indicate that this is necessary. It also attempts to have a maximum speed, a maximum density and account approximately for the four different signal types that are possible. The behaviour for this modified speed function is given in Figure 8



(a) *The imposed relation* $u(\rho)$. (b) $\rho(x)$

(c) $u(x)$ (d) $Q(x)$

Figure 2: *Evolution of the solution to the conservation equation, with* $u(\rho)$ *given by the equation above, at various time points. Equation solved numerically using Godunov's scheme with an initial distribution of* $\rho(x) = 1 - H(x)$. *Spatial domain is* $-150 \le x \le 150$, *and boundary conditions are* $Q(-150) = 0$, $\partial Q/\partial x(150) = 0$.

Figure 8:

3.4.3 Effect of Signals

(3.4.8) One possible extension to this continuum model was briefly considered. Here we might account for the train signals that are visible to drivers

indicating the density of trains further down the track. To model this, we can assume speed of a train to be a function of train density at a distance, d, ahead of the train, i.e.,

(3.4.9)
$$Q(x) = \rho(x)u(\rho(x+d)). \tag{10}$$

(3.4.10) We find this modification leads to unstable behaviour, when implementing this flux function within our numerical scheme. We believe this warrants further investigation. Additionally, as signals are placed at fixed points along the track, it may, in fact, be more accurate to model the speed as a function of density at discrete points

$$Q(x) = \rho(x)u(\rho(x_k)), \tag{11}$$

where $x_k > x$ is the location of the closest upcoming signal.

3.4.4 Conclusions

(3.4.11) This investigation seems to show general effects of modifying the train 'flux' that may be useful in predicting behaviour of train traffic and delays. An accurate measurement of the flux would allow us to test and refine predictions, so we would recommend exploring this further. Additionally, the incorporation of looking ahead to signals appears to have some interesting instability issues.

3.4.5 Flux Modelling

(3.4.12) In determining the dynamics of trains is is crucial to understand how the "flux" of trains is altered by signalling and the geometry of the track. We draw on the ideas of traffic modeling and note that, on a single track system, the flux is simply the density of trains times the velocity of the trains. We now explore what this flux might be and how it might be altered. We start by exploring a deterministic model of the flux but we note that in practice the flux will vary considerably with driver behaviour, train type, rail conditions and other source of uncertainty. We start by considering a single block of track and ask what the flux along this is. As the train enters a block there are four possible scenarios:

- the signal it just passed was green
- the signal it just passed was double yellow
- the signal it just passed was yellow.

In each of these cases we assume there two measurable quantities for the block. Firstly: the 'transit time', Tt, the time after entering the block that the train takes

to pass the next signal and be in the next block and secondly:the 'decision time', Dt, the time after entering the block where the driver has seen the next signal and must react to it such as coming to a halt at a stop signal.

We will briefly discuss general properties that these two quantities might have but would expect the data currently collected to give a very good insight into what the real behaviour is.

To examine what the resulting flux is it is informative to consider a simple diagram that might be put on the train path graph (train position (measured in block number) versus time (in minutes)). To fill the train route in we can now note that when put on this diagram we must have certain blocks and certain parts of blocks empty in order for the train to observe the signals that will make it run at a steady rate through the first block..

Here are the three examples where we assume the signal colour the train driver sees as they enter the block is the same colour as signal when they leave

For a train travelling 'under green signals' we have Figure 9. For a train travelling 'under double yellow signals' we have Figure 10. For a train travelling 'under yellow signals' we have Figure 11. From these we can determine the 'flux' of trains under each of the signal conditions. For example under green, using the Tt and Dt values for the bock under green, there is the block currently filled by the train which is occupied for Tt. Furthermore the two blocks ahead must also be empty for Tt and the final block that must be empty for at least $Tt - Dt$ so that the signal is green when it is observed by the driver. The total track used is therefore $4Tt - Dt$ block-minutes, hence the flux is $1/(4Tt - Dt)$ train/block-minute for a train under green. Similarly we have $1/(3Tt - Dt)$ train/block-minute for trains under double yellow and $1/(2Tt - Dt)$ train/block-minute for train under yellow. Critically, note that both Tt and Dt will have different values for each of the signal conditions and, in particular that they will be smallest under green because the train travels fastest and increase monotonically as the severity of the signal increases and the train travels slower. (Note we expect $Tt - Dt$ to also increase monotonically.)

What is not so obvious is which of the three possible fluxes is maximum. This is important since the maximum flux would give the fastest recovery from a disruption from the standard timetable. This is similar to the active speed limits on motorways where speeds are set to give maximum flux along the motorway. An interesting study would be to determine the three possible fluxes for all the blocks on a particular line and hence to find the blocks where the maximum fluxes are smallest and hence might indicate regions that will cause disruption to spread.

We can now consider using the existing timetable and the current status to create a new timetable by altering the path in the 'block-time' diagram while keeping the relevant additional blocks empty. We can also modify the shape of the path by removing the usual assumption of 'travelling under green' to one where part of the path is travelling under 'double yellow' or 'yellow'. Such alterations allow the steepness of the path to be reduced while simultaneously reducing the width of the

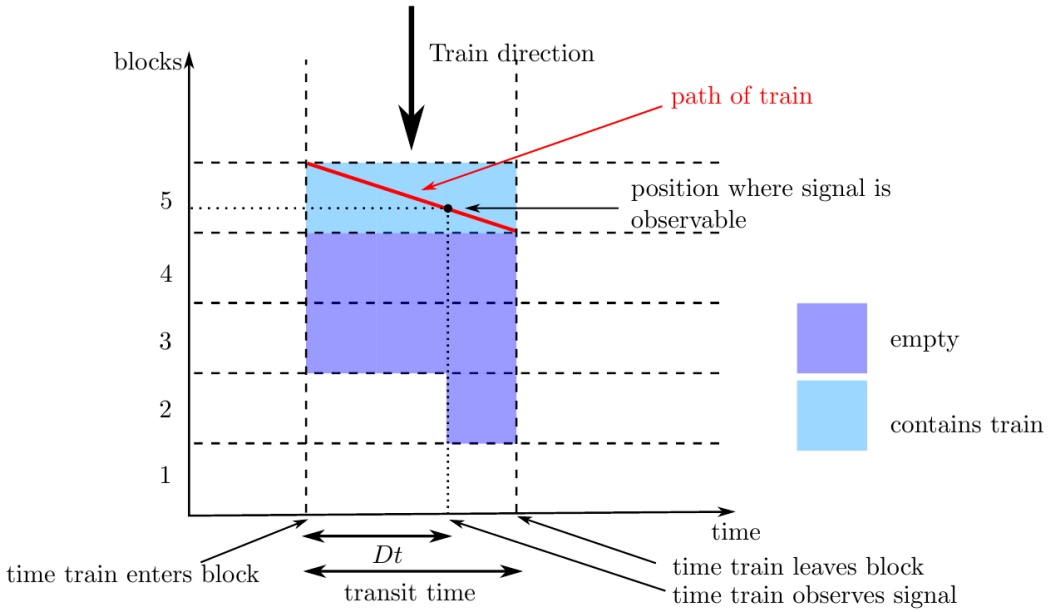


Figure 9: we assume the signal colour the train driver sees as they enter the block is the same colour as signal when they leave.

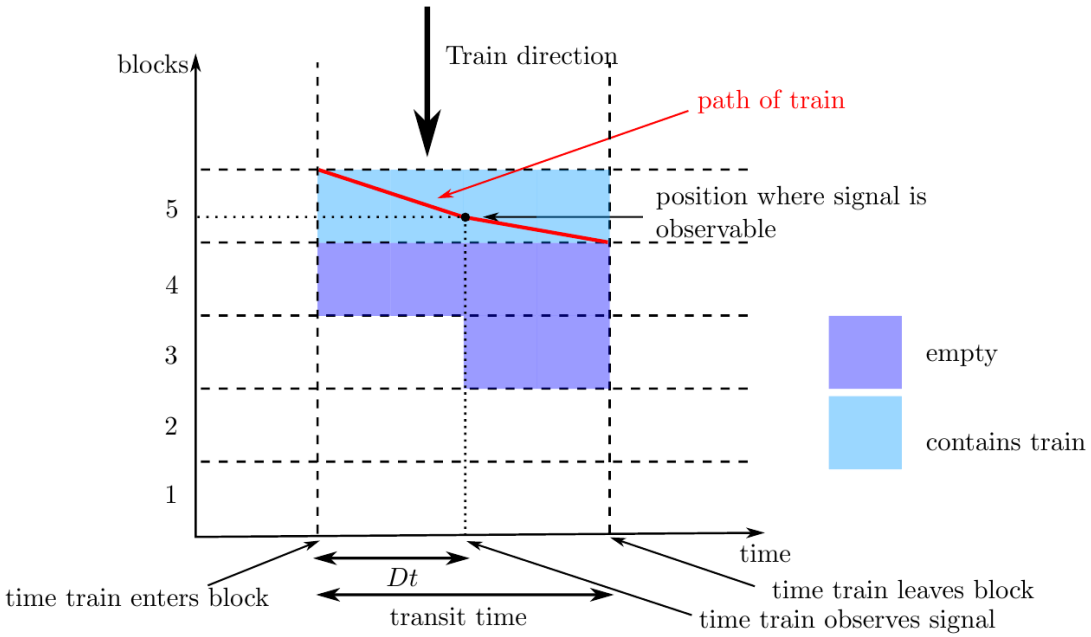


Figure 10: we assume the signal colour the train driver sees as they enter the block is the same colour as signal when they leave

path due to needing fewer blocks free around it.

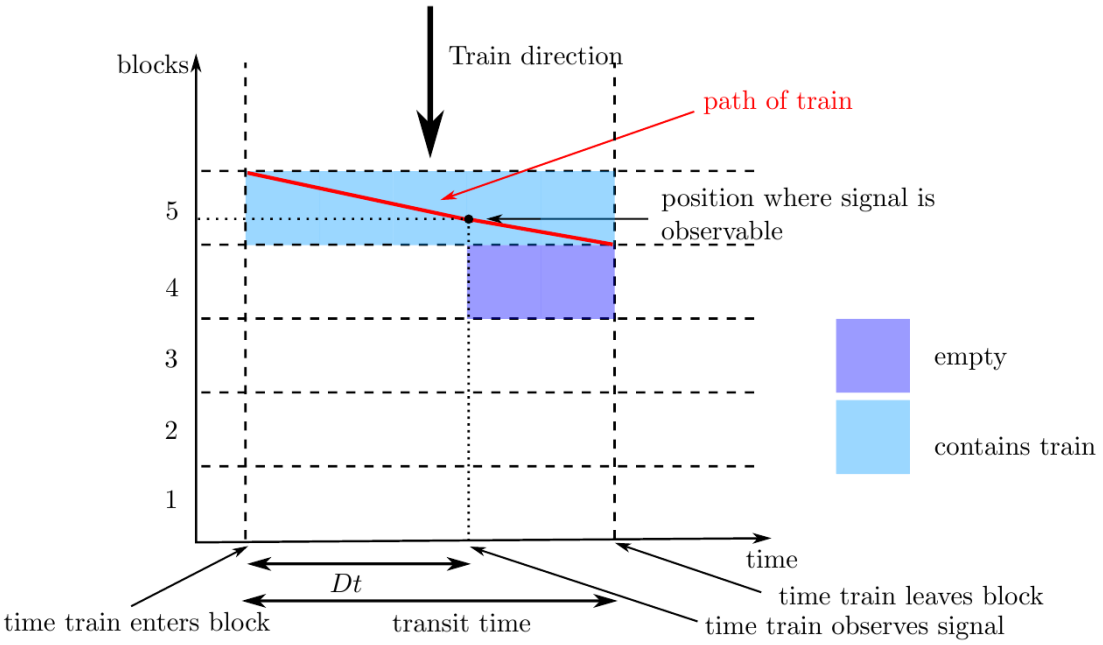


Figure 11: we assume the signal colour the train driver sees as they enter the block is the same colour as signal when they leave

3.4.6 Conclusion

(3.4.13) We have outlined a method for assessing possible parts of the track where difficulties may occur by examining the flux along the track. We have also described a method for graphically modifying a timetable that can accommodate understanding of this changing flux. Critical to such an assessment is to determine the dependency of transit time and the decision time on signal state and this requires looking in details at data.

(3.4.14) **Contacts:** Caoimhe Rooney, Jessica Williams, Colin Please

3.5 Approach 4: Predictive Modelling

3.5.1 Method A: Data driven method

(3.5.1) Aim to analyse the relationship between current operational behaviour (including current decision-making approach to deal with conflicts) and the quality of service (QoS) based on historical data.

(3.5.2) Given a fixed schedule, we use incremental delays at different stations to predict a specific QoS performance measure. Incremental delays at a station computed based on the scheduled arrival time depend only on what happens at the previous station/junction and on the track between

the two, and operational decisions made before the train arrives at that station, which can be used as inputs.

(3.5.3) There are several relevant QoS measures such as:
Total cumulative delay at a main station, e.g., Paddington
Average journey time between two main stations compared to the scheduled one
Average journey time of a typical customer on a common route. In order to compute this measure, we might need additional information. For example, one need to change trains to reach the destination on those common routes.

(3.5.4) **Implementation** The main dataset for this approach is the train movement dataset. Incremental delays can be computed from the actual delays recorded in the dataset. Given a train i and a station j, the incremental delay $\Delta d_{i,j} = d_{i,j} - d_{i,pred(i,j)}$, where $d_{i,j}$ is the actual delay of train i at station j, and $pred(i,j)$ is the previous station on the route of train i towards station j. For the origin station j_0, $\Delta d_{i,j_0} = d_{i,j_0}$.

(3.5.5) We start with a simple network shown in Figure 12 which consists of 4 stations: PADDTON (Paddington), HTRWAJN (Heathrow Airport Junction), HTRAPT (Heathrow Airport Terminals 1, 2, and 3), and RDNGSTN (Reading Station). There are two directed routes we would like to consider:
RDNGSTN $-->$ HTRWAJN $-->$ PADDTON and
HTRAPT $-->$ HTRWAJN $-->$ PADDTON.

(3.5.6) The QoS measure considered is the total cumulative delay at Paddington station. We would like to determine how delays happening in the network in a *one-hour period* affect the total delay at Paddington station in the *next one-hour period*. For each one-hour period, incremental delays for all trains within the network can be computed for all stations. If a train has not used a station in the network within that one-hour period, the delay is set to 0. The total cumulative delay at Paddington station can also be computed using the delays of all trains which arrived at Paddington within that one-hour period. Given the prepared dataset, predictive modeling can be applied using appropriate software such as R, IBM SPSS Modeler, and SAS Enterprise Miner.

(3.5.7) TO DO:
Extract necessary data from the main movement dataset for this simple network.
Analyse the results of the predictive model, especially the importance of each incremental delay in predicting the selected QoS measure.

(3.5.8) **Discussion** Are incremental delays good enough to represent the primary source of delays?

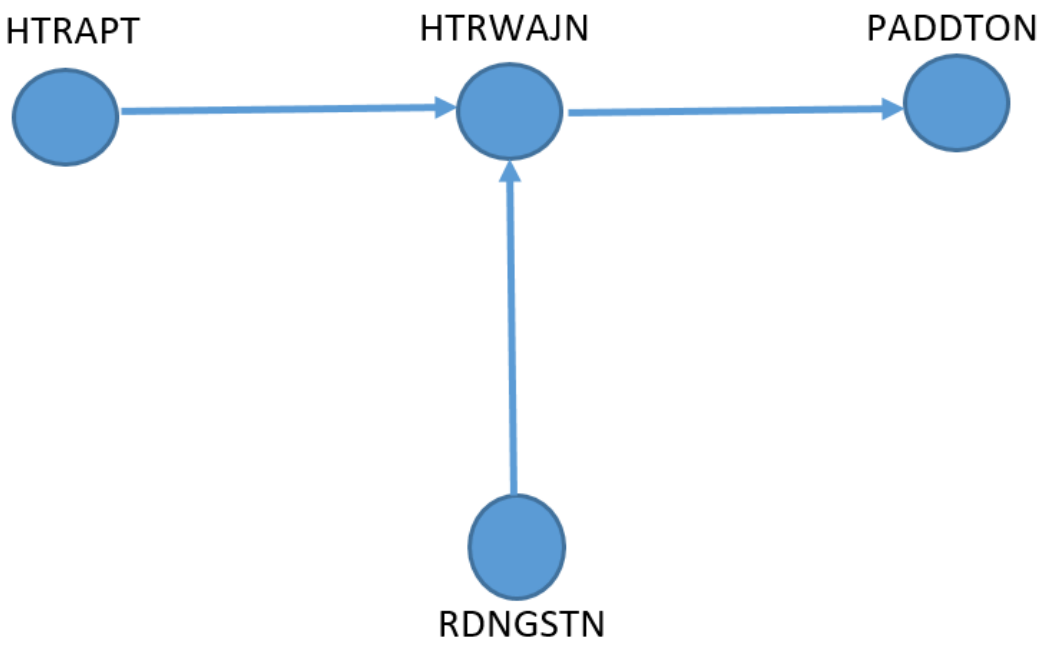


Figure 12: We start with a simple network which consists of 4 stations and consider two directed routes: Reading Station $--> $ Heathrow Airport Junction $--> $ Paddington and Heathrow Airport Terminals 1, 2, and 3 $--> $ Heathrow Airport Junction $--> $ Paddington.

We could try to predict the service quality at important stations (instead of overall QoS) by using relevant incremental delays from upstream (sub-)network. This approach could address the above issue of primary source of delays. Would it be enough to use the total incremental delay at each station in the network within a one-hour period to predict the delay at the main stations in the next one-hour period, i.e., removing the reference to particular trains in the inputs of the model?
In addition to analyse the relationship between current delays and QoS, it would be interesting to use historical data to analyse the uncertainty in primary sources of delays, e.g., traveling time between stations, delays caused by passengers at stations, driver availability, etc. These analyses can be useful in data preparation for the optimization models to construct the (*robust*) train timetable and to (*proactively*) reschedule the train timetable while taking into account potential delays in the future.
Assuming the delays at the main stations (e.g., Paddington station) can be estimated, an interesting problem would be how to reassign arriving trains to appropriate berths to make sure the delay in the future will be minimized. The problem would provide a *global* solution, once every half an hour, let say, while taking into account the anticipated delays of arriving trains.

3.5.2 Method B: Event based delay propagation in a train network

(3.5.9) **Introduction** In addition to the data driven method above, trains in networks can be characterized by logical relations: for example, train T_1 is in front of train T_2 on the same track segment, and at intersections it is decided which train will go ahead of another train. This implies that delays will propagate according to these logical relations. If train T_1 is behind train T_2 which is delayed then the delay may be passed onto train T_1. Similarly, if a train has to wait at an intersection for a delayed train then the delay is passed on. (Note: could also use a contagion model.

(3.5.10) The idea is to calculate the delays $d(k) = (d_1(k), \ldots, d_N(k))$ for trains $T_1, \ldots, T_N$ after event k for given initial delays $d(0)$. More precisely, we want to find functions f_k which correspond to individual events such that

$$d(k) = (f_k \circ \cdots \circ f_1)(d(0)). \tag{12}$$

(3.5.11) To illustrate the approach, let us consider a simple example. This likely relates to existing algorithms, for example: A delay propagation algorithm for large-scale railway traffic networks; doi:10.1016/j.trc.2010.01.002

3.6 Approach 5: Identifying the biggest contributors of delay in the network

(3.6.1) **The problem.** It is not only delay somewhere in the network that generates delay somewhere else in the network. It could be that a schedule is simply practically incompatible with the smooth operation of the rest of the network. One can imagine an ideal situation where a line is always on time and is olivious to the chaos around it. Perhaps having the entire schedule of this line shifted by (say) one minute could reduce a global measure of delays throughout the network. In other words, a train being on time can be bad for other trains anywhere in the network.

(3.6.2) An example: minimising delay at Paddington Station: One possible goal is therefore to identify which service and at which station is generating the largest amount of delay at Paddington Station. (Variations include the total amount of delay in the most congested cities in the network or even the entire network.) Delay at Paddington Station at a given time can be caused by any train in the network in the past. (One can model that the effects are felt only with the last τ units of time, e.g. three hours.) The symbol ω will very crudely represent realisations. (For example, if a service runs every day from 1 January to 31 December, then ω is simply a label of all the days of the year.) Let $R^\omega(t)$ be a train scheduled to leave Reading station at time t on a certain service (line) on realization ω. The delay of $R^\omega(t)$ is denoted

$$\delta_{R^\omega(t)}\,.$$

On the other hand, one measures the total delay at Paddington over a certain period of time, e.g.

$$\delta_{P^\omega}(t) := \sum_i \int_t^{t+\tau} \delta_i^\omega(s)\, ds\,,$$

where $\tau > 0$, where i indexes all services scheduled to arrive at Paddington during t and $t+\tau$, and $\delta_i^\omega(s)$ is the delay at time $t \le s \le t+\tau$ at Paddington Station of service with index i, on realisation ω. Other metrics are evidently possible. One then measures the correlation between δ_{R^ω} and $\delta_{P^\omega}(t)$. It may be that (on average) a delay for R of (say) 3 minutes will minimize the delay at Paddington. This suggests that the schedule of R should be modified accordingly. The above should be done for all services: all trains going through all stations in the network (including Paddington!) over all times.

(3.6.3) **More sophisticated metrics?** The above is evidently a very simple metric. Even keeping the same measure of delay at Paddington, more elaborate schemes of correlations involving several services should be investigated.

(3.6.4) **The use of deep-learning algorithms.** Deep learning algorithms have proved very powerful in identifying correlations. Examples are given in the book (for a general audience) *Big data: a revolution that will transform how we live, work and think, by Viktor Mayer-Schönberger and Kenneth Cukier.*

3.7 Approach 6: A Bayesian model of route status

(3.7.1) A Bayesian model looking at the posterior distribution of the health status at each meaningful interval along a route, given the delays incurred at each interval.

3.7.1 Problem to address

(3.7.2) Delay information of a train when it reaches certain points, or berths, on the route is available from the train movement data. However, it is difficult to know if a lateness is due to normal operational behaviour or if some unpredictable events might have happened. And, even if we knew when and where an action has had been taken to resolve a failure, the failure might have stemmed from points way before it was getting manifested.

3.7.2 Description of model

(3.7.3) We consider that there is a set of points distributed along the route, and we

observe certain amount of delays associated with each interval; the length of delay in any point segments, call it *relative lateness*, is dependent on what type of point the train is heading to.

(3.7.4) In this toy model we assume there are three types of points: junctions, stations and terminal (destination). We could fit a Gamma distribution to each of the point type, using the lateness data corresponding to the types, because a junction may have less lateness than a station, which may have less lateness than the destination.

(3.7.5) As an example, suppose the train is en route A—B—C—D, where B is station, C is junction and D is the destination of the train. We are interested in the lateness incurred between segments AB, BC and CD, which we could from observed historical data fit a distribution and it may be the case that on average the relative lateness at C is less than that at B which is less than that at D.

(3.7.6) The total delay adds up all the relative delays along the route, this constitutes one of the possible contingencies arising from normal operational behaviour. However unpredictable events can happen which cause further delays—the so-called primary delays. To model this kind of events, we also associate each interval with a status rating that ranges from 0 to 1, with 0 being normal and 1 being failure, the status rating of an interval measures the likelihood of primary delay in that interval.

(3.7.7) Suppose we have a route $0 \to 1 \to \ldots \to n$, where 0 is origin and n is destination, we assume train leave on time at origin, there are $n-1$ intervals. We introduce the following notations: T_i: relative lateness at interval $i, i \in \{1, \ldots, n\}$
T_n: relative lateness at the terminal, which is the final destination of the train.
D: $D = (D_1, \ldots, D_n)$ the collection of status ratings at each point, continuous random variable taking values from 0 to 1.
J, S: subset of $\{1, \ldots, n\}$ specifying the indices of junction and station points, respectively, such that $|J| + |S| + 1 = n$.

(3.7.8) Posterior distribution of status rating at each point given the relative lateness at the points:

$$\begin{aligned} P(D_1 \ldots D_n | T_1 \ldots T_n) &\propto P(D_1 \ldots D_n) L(T_1 \ldots T_n | D_1 \ldots D_n) \\ &= \prod_{i=1}^{n} P(D_i) \prod_{i=1}^{n} f(T_i | D_i) \end{aligned} \tag{13}$$

where $P(D_i)$ is the prior for the rating at i, that encodes our prior knowledge of tendency of primary delay during segment $(i-1, i)$, a Beta distribution can be used, with Beta(1,1) being the uniform distribution, indicating a vague prior.

(3.7.9) The term $L(T_1 \dots T_n | D_1 \dots D_n)$ is the likelihood of observing the relative lateness, given the status ratings. Here independence is assumed, that is relative lateness at i only depends on the rating at i. In a simple setting the conditional random variable $T_i|D_i$ is modeled as sum of two exponential random variables: $T_i|D_i = E_{1i} + E_{2i}$, corresponding to two lateness contributions: one from normal operational delays and is dependent on if i is in J, S, or terminal; the second is the contribution due to primary delays. In this model we use $E_{1i} \sim Exp(\mu_i)$ with

$$\mu_i = \begin{cases} \mu_J, & \text{if } i \in J \\ \mu_S, & \text{if } i \in S \\ \mu_T, & \text{if } i = n \end{cases}$$

and $E_{2i} \sim Exp(\lambda(D_i))$ with $\lambda(D_i) = 1/D_i - D_i$

(3.7.10) An MCMC simulation has been run to sample from the posterior distribution of the ratings, using *artificial* data. The code in R, as well as parameter settings, are listed in Appendix A.4. Figure 13 shows the posterior distribution of the ratings for each section, for the simulation above. Compare this with the observed relative lateness T_i, while we may expect large T_i leads to large D_i, there could be subtle cases where this isn't so obvious, and it would be interesting to investigate them further.

3.7.3 Further development

(3.7.11) This simulation could be run to sample from suitable real data.

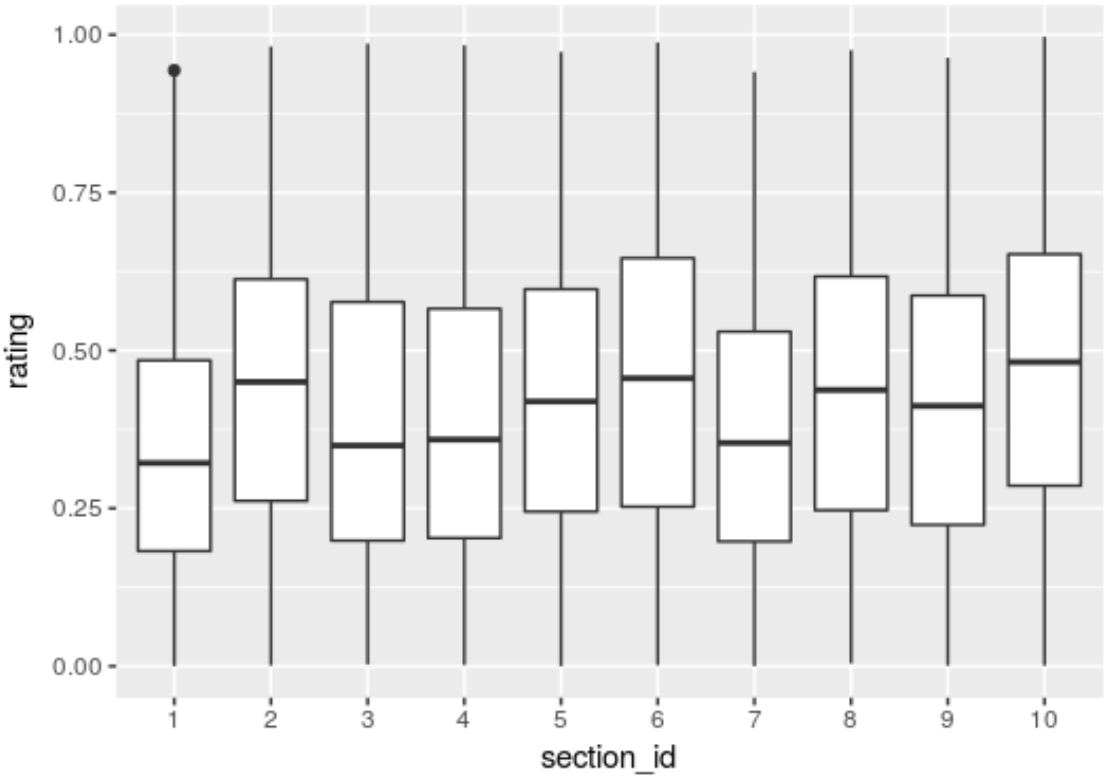


Figure 13: This plot shows the posterior distribution of the ratings for each section, for the simulation.

3.8 Approach 7: Bayesian Networks and IDSS

(3.8.1) In today's ever more interconnected world, decision making in dynamic environments is often extremely difficult despite vast streams of data, huge models and ever-growing disparate domains of expertise. Decision support can be valuable, but needs to incorporate all the relevant inputs in a coherent, transparent way so that decision makers can make defensible policy choices. In dynamic, plural environments decision makers often need a tool that can draw together expert judgements coming from a number of different panels of experts where each panel is supported by their own, sometimes very complex, models. Methodology and theoretical developments to do this has been recently developed [1].

(3.8.2) Complex networks of railways are one such example. How can we leverage these developments to increase efficiency through strategic decision-making? We first produce a probabilistic model of a single line, from Bristol to Paddington as a Bayesian Network, learn from data the conditional probability distributions and use this to understand how current operational behaviour, in terms of delays, affect service quality.

3.8.1 Method

(3.8.3) We represent the line from Bristol Temple Meads to London Paddington (Figure 14) as a Bayesian network (Figure 15). We then learn the probability distributions from data, and use domain knowledge to define sensible discretisations in the delays.

(3.8.4) In Figure 15 and following we discretise delays into ***No Delay*** (negative delays in freight trains which can leave early to zero delay); ***Minor delay*** (less than 10 minutes); ***Moderate delay*** (10.1-29.9 minutes); ***Severe delay*** (30 minutes plus, this is when compensation starts to be paid)

(3.8.5) We extract data into the format of Table 3.8.1 to learn the conditional probability distributions in Figure 16.

Journey	Station1	Station2	...
Journey 1	Delay1	Delay2	...
Journey 2	Delay1	Delay2	...

3.8.2 Results

(3.8.6) Note that most stations have minor delay as the most likely outcome, with the probability of moderate delays growing as we proceed along the network, as might be expected. We investigate the effects of delays at

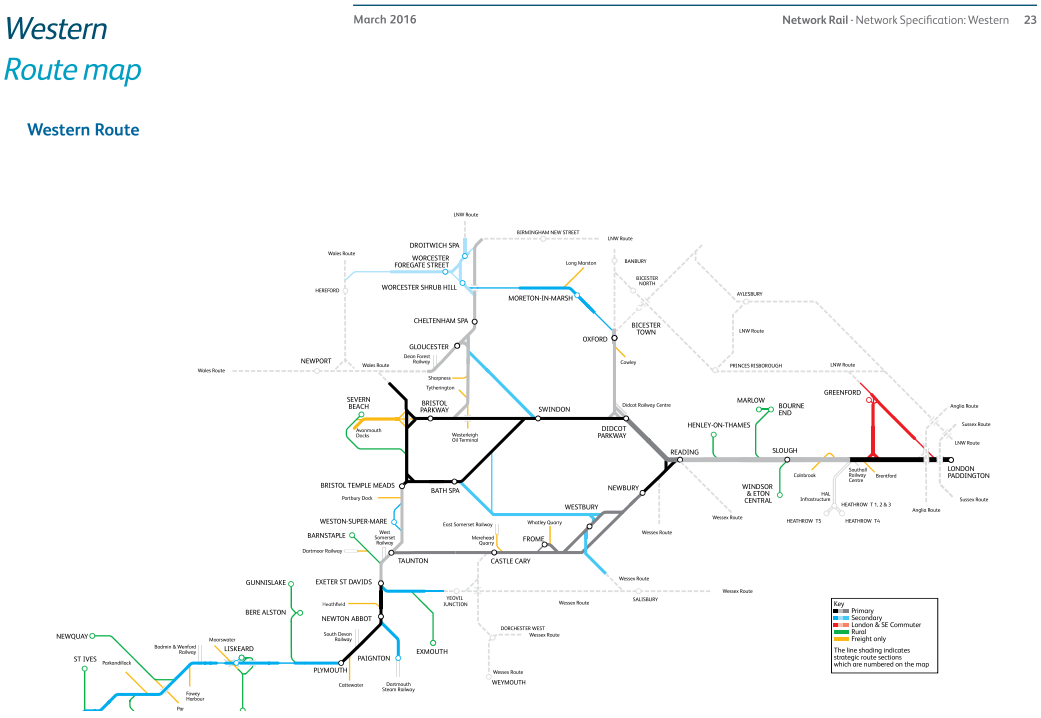


Figure 14: Route map

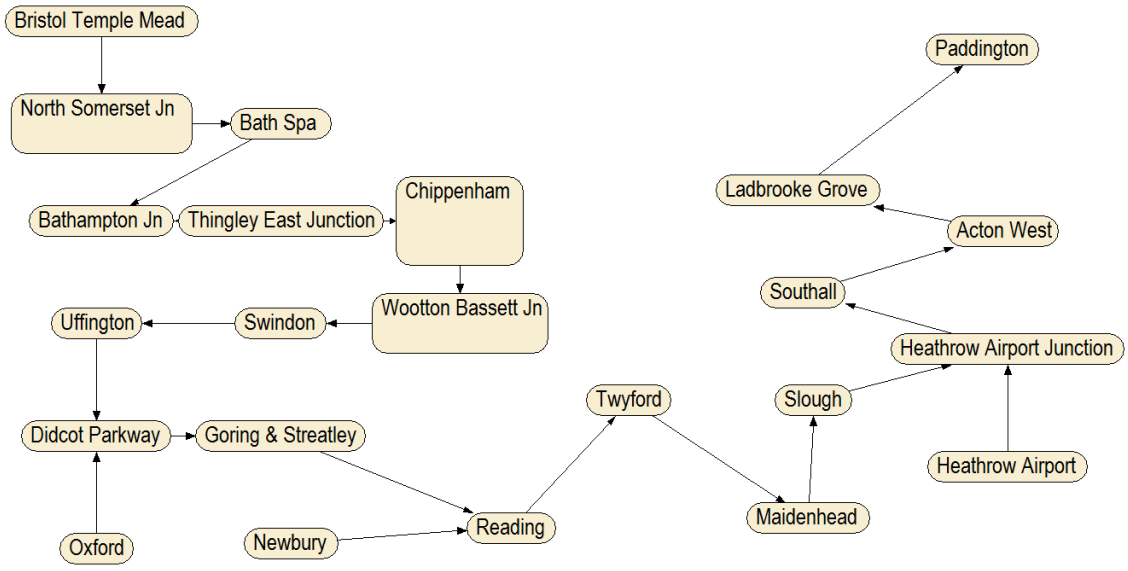


Figure 15: Bayesian network. Image produced in Netica [5]

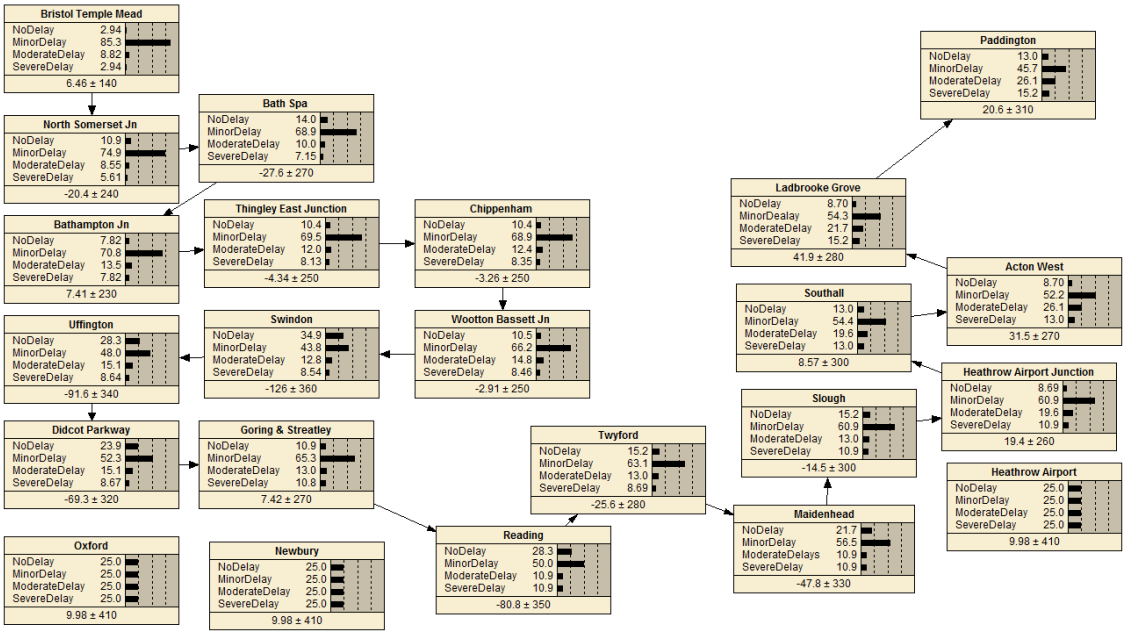


Figure 16: Bayesian network, with probability distributions learned from data. We then add key additional connections to the route (Oxford, Newbury,& Heathrow Airport), initially with flat prior probabilities. Image produced in Netica [5]

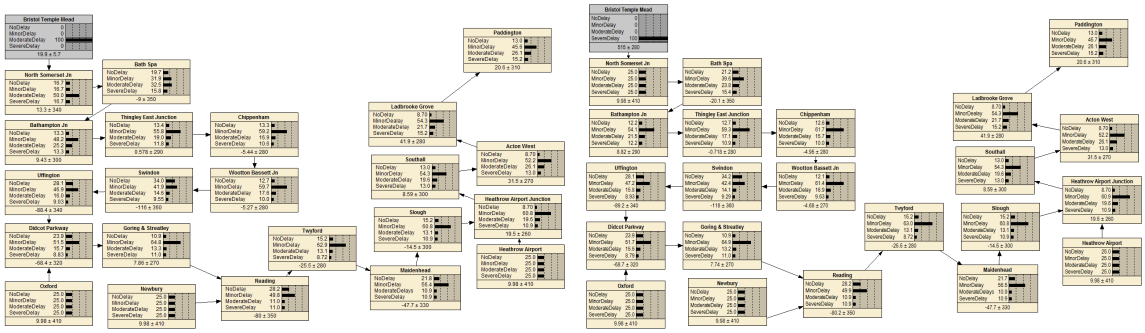


Figure 17: Bayesian network, with probability distributions learned from data, showing the downstream effects on delay probability distributions of moderate (left) and severe (right) delays at Bristol Temple Meads. Image produced in Netica [5]

upstream stations aon Paddington using a what if analysis

(3.8.7) A what-if analysis shows that a moderate delay or severe delay at Bristol Temple Meads is likely to have dissipated by Didcot Parkway, with no effect at Paddington, Figure 17. A moderate delay or severe delay at North Somerset Junction, Bath Spa or Bathampton Junction is likely to have dissipated by Didcot Parkway, with no effect at Paddington. A moderate delay or severe delay at Thingley East Junction or Chippenham is likely to increase slightly the probability of moderate delay at Didcot Parkway,

with no effect at Paddington. A moderate delay or severe delay at Chippenham is likely to increase slightly the probability of moderate delay at Didcot Parkway, with no effect at Paddington. A moderate delay or severe delay at Wooton Bassett Junction is likely to increase the probability of moderate delay at Didcot Parkway, with no effect at Paddington. A moderate delay or severe delay at Swindon is likely to increase significantly the probability of moderate delay or severe delay at Didcot Parkway, with no effect at Paddington. A moderate delay at Uffington is highly likely to lead to moderate delay at Didcot Parkway, with no effect at Paddington. The effect of a severe delay is a flat distribution at Didcot, suggesting this combination has not been seen in the data. Moderate or severe delays at Southall, Acton West or Ladbrooke Grove are highly likely to lead to moderate or severe delays at Paddington, as might be expected, given their proximity and the limited opportunity for corrective interventions.

(3.8.8) Another approach is to assume that a severe or moderate delay has been observed at Paddington and examine the probability distributions at the upstream stations. Figure 18 and Figure 19 show that a moderate or severe delay at Paddington implies an upstream delay beginning at Heathrow Airport Junction and propagating though to Paddington. Further investigations could be made if data on delays at Heathrow Airport, Oxford and Newbury ere obtained or those same journeys and the probability distributions on delays given delays at multiple upstream stations could be investigated.

(3.8.9) We can use subjective or heuristic probabilities for the conditional probability tables, where no data exists, and estimate the effects of delays at two feeder stations on the following stations, shown in Figures 20, 21, 22.

(3.8.10) This provides a probabilistic model for a single journey. Similar models could be developed for whole regions, and these networked together using principles in [1]. If a suitable multi-attribute utility can be elicited from decision-makers, which has measurable attributes and passes the clarity test, then the networked probabilistic models for regions along with other key expert panels (e.g. weather) can be used to evaluate candidate policy options and score each with respect to the utility defines, and taking uncertainty into account. Figure 23 gives a sketch of such an integrating decision support system.

3.8.3 Discussion

(3.8.11) The main limitation of this approach is the limitations within the data. The delay timed recorded are actual delay times, after any mitigating policies have been applied. We have no record of what mitigating policies were applied nor where they were applied along the journey.

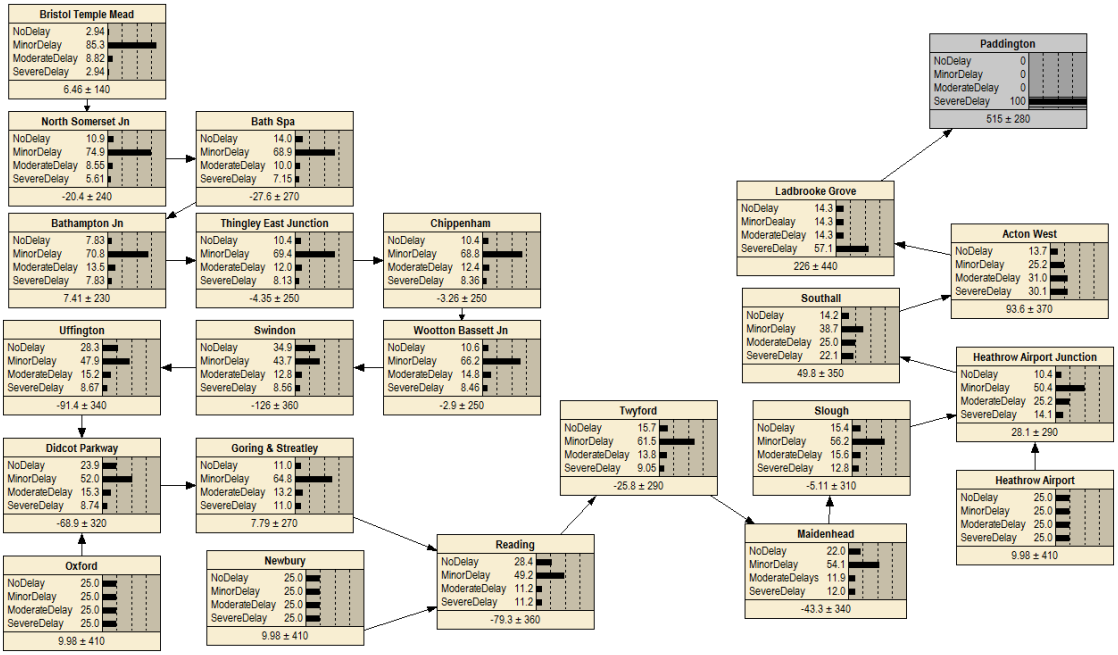


Figure 18: Bayesian network, with probability distributions learned from data, showing the upstream effects on delay probability distributions a severe delay into Paddington. Image produced in Netica [5]

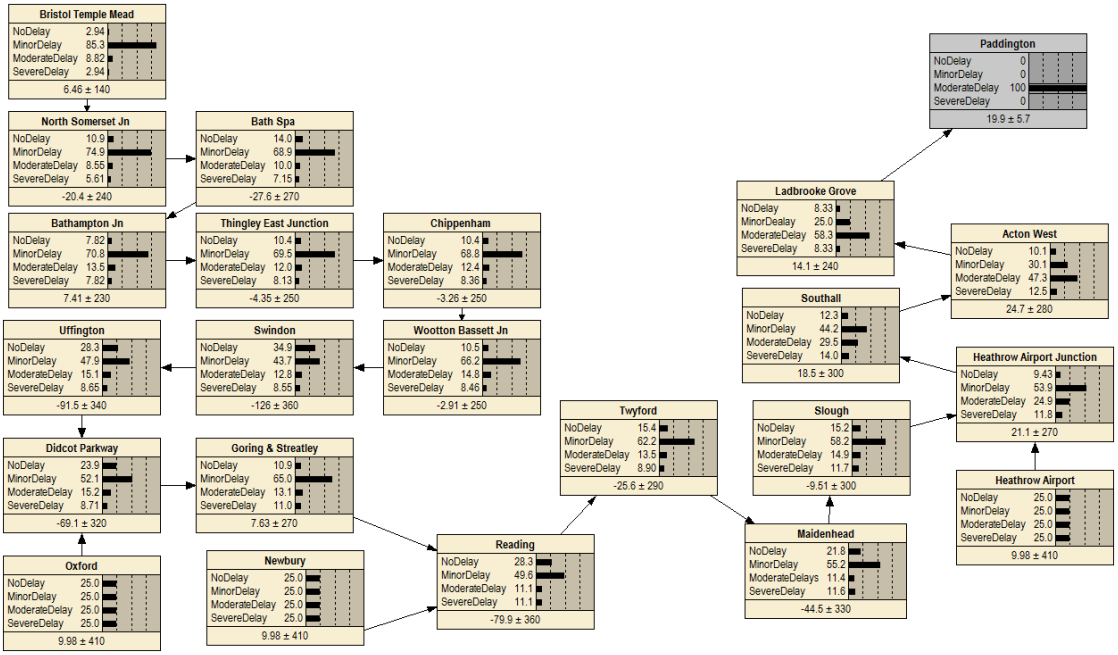


Figure 19: Bayesian network, with probability distributions learned from data, showing the upstream effects on delay probability distributions a Moderate delay into Paddington. Image produced in Netica [5]

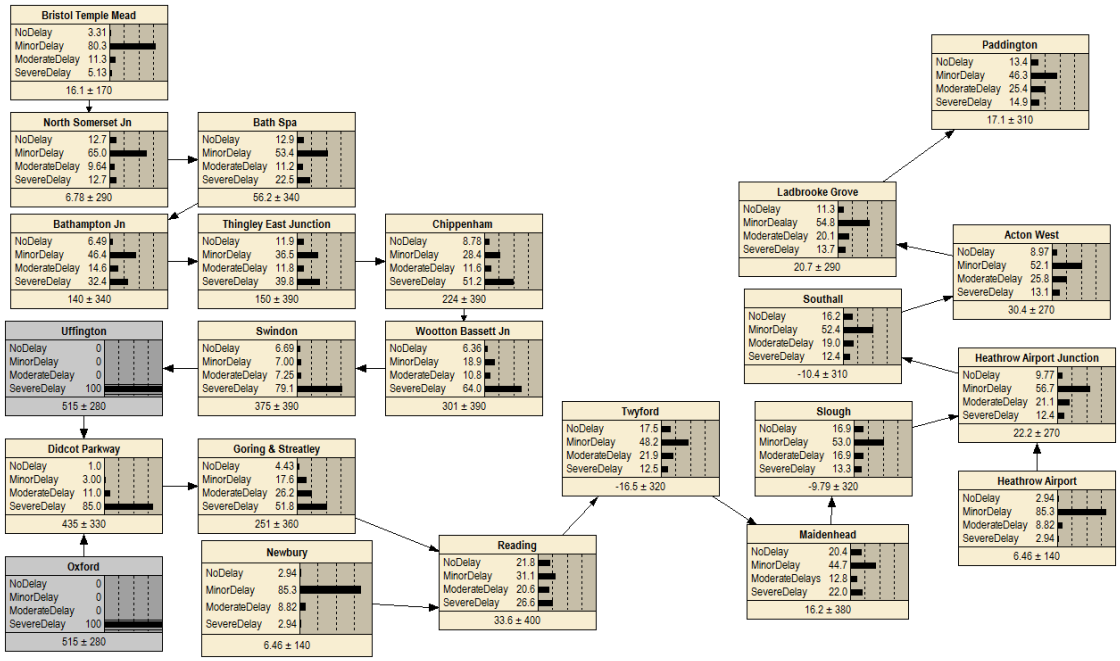


Figure 20: Bayesian network, with probability distributions learned from data augmented with heuristic and subjective probabilities where data sparse or non-existent. This shows the downstream effects on Paddington of a severe delay at both Uffington and Oxford, both feeding into Didcot Parkway. Image produced in Netica [5]

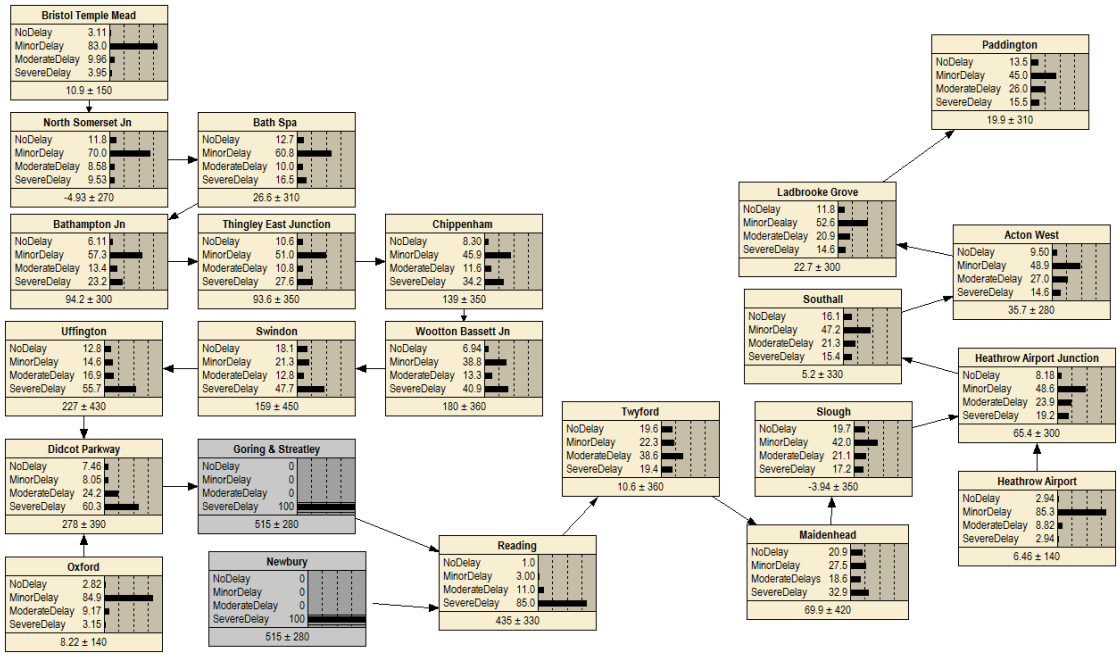


Figure 21: Bayesian network, with probability distributions learned from data augmented with heuristic and subjective probabilities where data sparse or non-existent. This shows the downstream effects on Paddington of a severe delay at both Goring & Streetley and Newbury, both feeding into Reading. Image produced in Netica [5]

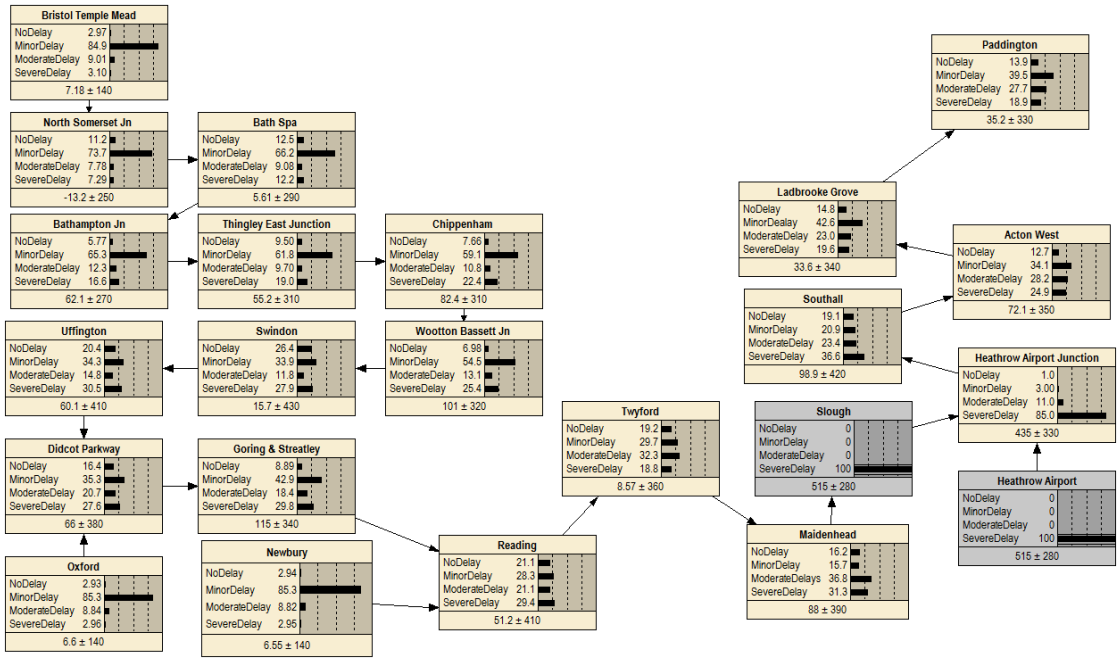


Figure 22: Bayesian network, with probability distributions learned from data augmented with heuristic and subjective probabilities where data sparse or non-existent. This shows the downstream effects on Paddington of a severe delay at both Slough and Heathrow Airport, both feeding into Heathrow Airport Junction. Image produced in Netica [5]

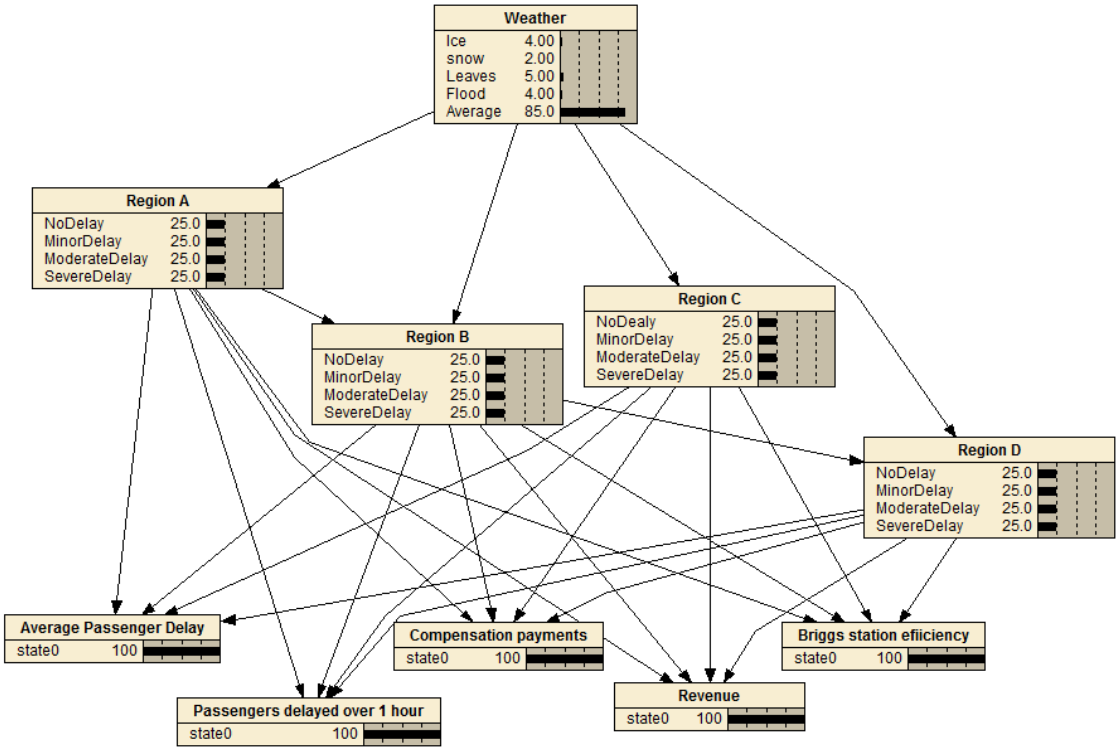


Figure 23: Sketch Integrating Decision Support System (IDSS) with 5 attributes in the Utility: average passenger delay, passengers delayed over one hour, compensation payments, revenue and Briggs station efficiency [11]. Weather can affect any region and all regions contribute to the utility function. Region A affects delays in Region B and Region B affects Region D but regions D is conditionally independent of Region A given Region B, i.e. if we know the status of Region B, knowing the status of Region A gives no further useful information to predict the status of Region D. Region C is independent of other regions. Image produced in Netica [5]

(3.8.12) There is now methodology to knit together probability models for different parts of a complex rail system for decision support. Called an integrating decision support system (IDSS) this makes possible coherent inference over a network of probabilistic models [1, 2, 3]. With this methodology, a decision making panel can define a utility function and the IDSS can be used to score candidate policies to aid selection. This approach is transparent giving decision-makers the ability to justify decisions to an auditor. Uncertainty can be propagated using Tower Rules $\mathbb{E}(X) = \mathbb{E}(\mathbb{E}(X|Y))$ and $Var(X) = \mathbb{E}(Var(X|Y)) + Var(\mathbb{E}(X|Y))$ (sometimes called the law of total probability for the expectation and conditional variance identity). This allows uncertainty to be incorporated into the score for candidate policies.

3.8.4 Future developments

(3.8.13) * Gather data on the interventions enacted to mitigate delays and estimate effect of *intervention v no intervention* perhaps by structured expert judgement approaches [6, 7, 8, 2]
* Represent the rail system as a set of probabilistic models in a way that is suitable to the domain, e.g. by management regions. Add in probabilistic information from any external influencing factors, e.g. weather forecast indicating ice / heat / snow / likelihood of leaves on the line.
* Develop a mulitattribute utility against which to evaluate candidate interventions [4], e.g. overall delay, number of passengers delayed more than x minutes, costs of compensation to passengers / train operators.
* Develop IDSS [1] to score candidate policies to provide decision support.

(3.8.14) **Contact:** Martine J. Barons

3.9 Approach 8: Mixed Integer Programming

(3.9.1) It's also a toy model of a sort. **Method** Aim to reschedule traffic in a limited time window (e.g., 60 minutes into the future) on a macroscopic level (that is, arrivals/departures to/from major stations and junctions; do not pay too much attention to detailed track topology and dynamic properties of rolling stock, etc.). The objective is to make the new schedule as close to the original one as possible.

3.9.1 Decision variables

(3.9.2) The rescheduled time of each *event*. An *event* is the arrival of a particular train in a particular location, or the departure of a particular train from

a particular location.
For certain pairs of events, the precedence relation. **Constraints**

- Open track capacity is modelled by *headway constraints*: minimum time between departures/arrivals of two trains to/from the same line.
- Overtaking: where overtaking is impossible (open track and certain stations/junctions), the precedence relation must be preserved.
- Train must not depart earlier than advertised time of departure.
- Minimum travel times between pairs of consecutive stations (depends on class of train, but can be refined to be more train-specific).
- Minimum dwell times in stations where the train stops (also depends on class of train, but can be refined to be more train-specific).
- Known actual departure times ('boundary conditions') – current system behaviour.

TO DO: How to model conflicts between arriving and departing trains at Paddington?

Objective Minimise total delay = sum of (rescheduled time of event - original time of event) over all advertised arrivals of trains.

Implementation Coded using the *Mosel* modelling language to be solved by *Xpress*. (Because JF has a licence for Xpress and no licence for Cplex or so.) See Appendix A.3 for code.

Results We have implemented the model in a simple situation of the Reading–Paddington line, with data for 7 trains. In simple scenarios it shows that sometimes it may be useful to reverse the order of running trains, sometimes it may be beneficial to let trains overtake at stations to deal with disruptions (overtaking is not really possible between Reading and Paddington, but we allow it in the model to test these aspects). In one scenario we can see how the delay of a train departing from Reading propagates to a train going back from Paddington to Reading (same rolling stock).
Discussion Strengths The model deals with disturbances on a more global scale by considering the impact of traffic-management decisions on the whole network. The solution is optimal with respect to preserving the original schedule as much as possible.
Limitations The model does not take into account detailed-level information, such as platforming, exact track topology, dynamic properties of rolling stock, etc. However, more constraints can be added if necessary to model some of these aspects. Scalability is yet unknown, but it appears that rescheduling an hour of traffic on the GW network should be feasible. The practicality of this approach has two parts: 1. Can we solve the model in reasonable computational time? 2. Is the resulting schedule feasible in practice? In practice, there will always be a trade-off between computational time and quality of new schedule.

Next steps (& Scalability)

- Test whether the model can be scaled up to usable size.
- Build a front end that will allow the user to study the effects of different traffic management decisions, and the propagation of various disruption scenarios.
- Integration with "timetable surgery" interventions?

Open questions What is the effect of "riding the yellows" on travel times? Can we really ignore it in the model? (See [Approach 2])

Contact: Jan Foniok

3.10 Approach 9: How to improve global outcome from local decisions?

(3.10.1) **Model of current implementation.** The entire network is subdivided into smaller portions. Each portion has a local controller which makes decisions. They actually see a slightly larger portion of the network which it controls. (There, the local controller cannot act on this extra portion.) The local controllers do not see or take into account what happens of this extended portion of the network in their decision-making process (algorithm).

(3.10.2) **A possible weakness of this implementation.** If a local controller sees all the berths free just outside the portion that they control, they will presume that is ok to allow all trains to leave this portion. This works to optimise the performance of the local portion of the network. On the other hand, the local controller does not take into account what happens outside this local portion, and it may be that allowing trains to leave this local portion based on the fact that the berths are unoccupied could have serious consequences somewhere else in the network.

(3.10.3) **The problem** How can information from outside of the local portion of the network be used in order to prevent decisions which are good locally from having bad consequences globally? How would this be implemented? In what form would information from the network be fed back to the local controller?

(3.10.4) **A suggestion:** A unique global controller has access to the entire network, and uses this global information to modify the local information used by the local controllers. The local controllers do not change their decision-making algorithms.

(3.10.5) **Feeding back global information to the local controllers.** Let P be a portion of the network controlled by a certain local controller. Let $P^{'}$ be the slightly larger portion of the network which this local controller sees and therefore uses to make decisions on traffic in P. Let D denote the set of rules used by this local controller. Again, D is used to acts on P, and uses the information on $P^{'}$. Let N denote the entire network. There is a unique global controller which sees the entire network N. This global controller decides what they want to optimise. The global controller does not directly act on the network, but is somehow able to influence the local controllers in such a way to achieve the global controller's optimisation problem - whatever it is. Here is one possible implementation of how the global controller is able to do this with regards to the local controller acting on the portion P. Based on what the global controller knows on the entire network N (e.g. current state or history), they generate a virtual portion V of the network in the following way. The portion V is the same as the portion $P^{'}$, except that they can overwrite the information on the subset of $P^{'}$ not in P. The local controller uses the same set of rule D, but applies them to V, rather than $P^{'}$, in order to act on P.

(3.10.6) How does the global controller decide what information to send to the local controllers? The global controller decides what should be minimised. The algorithm used by the global controller to generate the virtual local network V

4 Conclusions

(4.0.7) There are a number of promising approaches which provide useful lines of enquiry, demonstrating the strong potential of industry - academia collaborations. Many of the study group approaches are suitable for expansion beyond the simple railways modelled to include variable train speeds, junctions and intersections, temporal differences in usage, such as tidal flows in and out of cities, and resource constraints.

A Appendices

A.1 First appendix: Toy Model code in Python 2

https://codebunk.com/i/nN2ugbCqRS9FMDGTgPar

https://codebunk.com/i/nN2ugbCqRS9FMDGTgPar

Code in Python2:

```python
import numpy as np


#example of network_matrix
network_length=17
ad_matrix = np.identity(network_length)

ad_matrix[0][2] = 1
ad_matrix[1][2] = 1
ad_matrix[2][3] = 1
ad_matrix[3][4] = 1
ad_matrix[3][5] = 1
ad_matrix[4][6] = 1
ad_matrix[5][6] = 1
ad_matrix[6][7] = 1
ad_matrix[7][8] = 1
ad_matrix[8][9] = 1
ad_matrix[8][10] = 1
ad_matrix[9][11] = 1
ad_matrix[10][11] = 1
ad_matrix[11][12] = 1
ad_matrix[12][13] = 1
ad_matrix[12][14] = 1
ad_matrix[13][15] = 1
ad_matrix[14][15] = 1
ad_matrix[15][16] = 1


time_steps= 100
```

```
def generate_nodes(ad_matrix,network_length):
nodes=[0 for x in range(network_length+1) ]
#A vector containing all the nodes (as classes), the zero position are for the tra
for i in range(network_length):

#Here we check for each node all the edges that it reaches (including it self)
a=np.where(ad_matrix[i,:]==1)[0]
a = np.delete(a, np.where(a==i)[0])+1
#Here we check for each node all the edges that gets there (including it self)
b=np.where(ad_matrix[:,i]==1)[0]
b = np.delete(b, np.where(b==i)[0])+1

if len(a)==1 and len(b)==1:   #line case
kind=0
elif len(a)==1 and len(b)>1:    #split case
kind=2
elif len(a)>1 and len(b)==1:  #joint case
kind=1

elif len(a)==0:   #last node, we will send it to the 0 node
kind=4
nodes[0]=node(0,5,[i+1],[0])
a=[0]

else: #starting edge
kind=6
nodes[i+1]=node(i+1,kind, b, a)   #We leave the first one empty

#We still need to add stations, those are line nodes (0),
#which have a split node (1) as nodes_in and a joint node (2) as node out.
for i in range(network_length):
if nodes[i].kind==0 and nodes[nodes[i].nodes_in[0]].kind==1 and
nodes[nodes[i].nodes_out[0]].kind==2:
nodes[i].kind=3


return nodes




class node(object):
```

```
#We save all the nodes in a vector, the 0 one corresponds to the trains leaving

"""
Node class
Attributes:
position: cardinal number indicating the edge position
kind: type of node, could be 0-line , 1-split, 2-joint 3-station 4-end
                5-out 6-start (overtaking are taking place just here)

"""

def __init__(self, position, kind, nodes_in, nodes_out):
self.position = position
self.kind= kind
self.nodes_in= nodes_in
self.nodes_out=nodes_out


class train(object):
"""Here we construct the train class

Attributes:
starting_position: where the train starts.
starting_time: when it enters in the rail
ending_time: when it gets out from the rail
category: train category (linked to the path so far)
station_stop= a flag, it is one when it stops at a station, it turns 0
                                                        when it leaves
"""

def __init__(self, position, time, category, delay):
self.starting_position= position
self.starting_time= time
self.ending_time= 0
self.category= category
self.station_stop= 0 #flag
self.delay= delay
"""
def decision_move(node_to_update):
num_nodes_in = len(node_to_update.nodes_in)
if num_nodes_in == 1: # if the node is in a row of nodes
pos_train = node_to_update.nodes_in[0]
if trains_now[pos_train] != 0:
trains_now[node_to_update.position] = trains_now[pos_train]
trains_now[pos_train] = 0
```

```
else:
print(0)

"""
def ordering_nodes(ad_matrix, network_length):
final_list = []
ad_matrix_diag = ad_matrix.copy()
for i in range(network_length):
ad_matrix_diag[i,i] = 0

nodes_to_check = np.sum(ad_matrix_diag, axis = 1) == 0
checked_nodes = np.zeros((network_length), dtype=bool)
temp_ad_matrix = ad_matrix_diag.copy()

# Loop
for k in range(network_length):
if sum(checked_nodes) != network_length:
# if it is not empty
nodes_to_check = np.sum(temp_ad_matrix, axis = 1) == 0
positions_to_check = np.where(np.multiply(np.array(nodes_to_check == True),
                                          np.array(checked_nodes == False)))

#
for i in range(len(positions_to_check)):
temp_vect = []
temp_vect = [positions_to_check[i] + 1]
checked_nodes[positions_to_check[i]] = True
for j in range(i+1,len(positions_to_check)):
if ad_matrix[:,positions_to_check[i]] == ad_matrix[:,positions_to_check[j]]:
temp_vect.append(positions_to_check[j] + 1)
checked_nodes[positions_to_check[j]] = True
final_list.append(temp_vect[0])

temp_ad_matrix = ad_matrix_diag.copy()
temp_ad_matrix[np.where(checked_nodes == True),:] = 0
temp_ad_matrix[:,np.where(checked_nodes == True)] = 0

return final_list


#decision process with the line edges
def e_line(pos,transition,nodes):
# check if a train is already on the next spot, if not we move the train
                                        #to the next edge, otherwise it stops
if not nodes[pos].nodes_out[0] in transition:
```

```
transition[pos] = nodes[pos].nodes_out[0]
else:
transition[pos] = pos
return transition

#decision process with the split edges
def e_split(pos,transition,nodes):
# check if there is an empty edge in the next ones
flag=0
for t in nodes[pos].nodes_out:
if not t in transition:
transition[pos]= t
flag=1
#if it does not find any empty edges it stops there
if flag==0:
transition[pos]= pos
return transition

#decision process with the join
def e_join(list_pos, transition,trains,nodes):
#list_pos: position of the existing trains on the same level

if len(list_pos)==0:
return transition
# check if a train is already on the next spot, if yes we stop all the trains
next_node=nodes[list_pos[0]].nodes_out[0]
if next_node in transition:
for x in list_pos:
transition[x]= x
trains[x].station_stop=1
else:
L=[trains[x].category for x in list_pos]
#list of categories of trains at the station
if len(which(L,max(L)))==1:
if max(L) in [0]: # there may be more categories that need to stop so [0,1,2,...]
if trains[list_pos[which(L,max(L))[0]]].station_stop == 0:
trains[list_pos[which(L,max(L))[0]]].station_stop = 1
transition[list_pos[which(L,max(L))[0]]] = list_pos[which(L,max(L))[0]]
else:
trains[list_pos[which(L,max(L))[0]]].station_stop = 0
transition[list_pos[which(L,max(L))[0]]] = next_node
else:
transition[list_pos[which(L,max(L))[0]]]=next_node
trains[list_pos[which(L,max(L))[0]]].station_stop=0
list_pos = np.delete(list_pos, which(list_pos,list_pos[which(L,max(L))[0]]))
```

```
for x in list_pos:  #the others stay there
transition[x]=x
trains[x].station_stop=1
elif max(L)>0:
fastest_trains=[list_pos[t] for t in which(L,max(L))]
times=[trains[x].starting_time for x in fastest_trains]  #fastest=latest
winner=fastest_trains[which(times,min(times))[0]]   #The most delayed goes head
transition[winner]=next_node
list_pos = np.delete(list_pos, which(list_pos,winner))
trains[winner].station_stop=0
for x in list_pos:  #the others stay there
transition[x]=x
trains[x].station_stop=1
elif max(L)==0:
stopped_trains=[t for t in list_pos if trains[t].station_stop==1 ]
times=[trains[t].starting_time for t in stopped_trains]
winner=stopped_trains[which(times,min(times))[0]]   #The most delayed goes head
transition[winner]=next_node
list_pos= np.delete(list_pos, which(list_pos,winner))
trains[winner].station_stop=0
for x in list_pos:  #the others stay there
transition[x]=x
trains[x].station_stop=1

return transition

def e_enter(list_pos, transition,trains,nodes):
if not list_pos:
return transition
else:
# check if a train is already on the next spot, if yes we stop all the trains
next_node=nodes[list_pos[0]].nodes_out[0]
if next_node in transition:
for x in list_pos:
transition[x] = x
else:
L=[trains[x].category for x in list_pos]
#list of categories of trains at the station
if len(which(L,max(L)))==1:
transition[list_pos[which(L,max(L))[0]]]=next_node
list_pos= np.delete(list_pos, which(L,max(L))[0])
for x in list_pos:  #the others stay there
transition[x]=x
else:
fastest_trains=[list_pos[t] for t in which(L,max(L))]
```

```
times=[trains[x].starting_time for x in fastest_trains]  #fastest=latest
winner=fastest_trains[which(times,min(times))[0]]
#The most delayed goes ahead
transition[winner]=next_node
list_pos= np.delete(list_pos, which(list_pos,winner))
for x in list_pos:  #the others stay there
transition[x]=x
return transition
```

```
def new_move(trains_now,nodes,ord_nodes,network_length,
                    prob_stop_anywhere,prob_stop_station):
transition=[0 for x in range(network_length+1) ]
#we initialize the transition vector, then we fill it following the
                                #ord_nodes which starts from the bottom
for curr_pointer in ord_nodes:
list_pos=[]
for i in curr_pointer:
curr_train=trains_now[i]
if curr_train: #it is true if there is a train
list_pos= list_pos+[i]
if len(list_pos)==0:  #means there are no trains here
continue
else:
node_kind=nodes[list_pos[0]].kind  #they have all the same kind
if node_kind==0 or node_kind==2:  #Just one node here
if np.random.rand() < prob_stop_anywhere:
transition[list_pos[0]] = list_pos[0]
continue
else:
transition=e_line(list_pos[0],transition,nodes)
elif node_kind==1:  #Just one node here
if np.random.rand() < prob_stop_anywhere:
transition[list_pos[0]] = list_pos[0]
continue
else:
transition=e_split(list_pos[0],transition,nodes)
elif node_kind==3:
for i in list_pos:
if np.random.rand() < prob_stop_station:
```

```
transition[i] = i
trains_now[i].station_stop=1
list_pos = np.delete(list_pos,which(list_pos,i))
transition=e_join(list_pos, transition,trains_now,nodes)
elif node_kind==4:   #we can change that later if we want cause delays
if np.random.rand() < prob_stop_anywhere:
transition[list_pos[0]] = list_pos[0]
continue
else:
transition[list_pos[0]]=0
elif node_kind==6:
transition=e_enter(list_pos, transition,trains_now,nodes)
return transition


#We take out from our system the trains i    for x in list_pos:f they end their pa
def store(trains_now,trains_old, time):
train_exit=trains_now[0]
if train_exit: #if there are trains in the exit position
train_exit.ending_time=time
trains_old.add(trains_now[0])
trains_now[0]=0
return trains_now, trains_old

def new_train(pos,trains_now,time,category,train_delay):
#before check that the position it's empty
trains_now[pos]=train(pos,time,category,train_delay)
return trains_now

def refresh(trains_now,transition,network_length):
#refresh the trains_now list
new=[0 for x in range(network_length+1)]
for t in range(network_length+1):
new[transition[t]]=trains_now[t]
return new


#main function
def railtrack(ad_matrix,time_steps,prob_inject,lambda_delay,
                          prob_stop_anywhere,prob_stop_station):
network_length = ad_matrix.shape[0]
#first we generate the nodes vector:
nodes=generate_nodes(ad_matrix,network_length)
#We order the nodes by priority for making decision,
                 #starting from the end of the rail
```

```
ord_nodes=ordering_nodes(ad_matrix, network_length)
#Here we initialize the trains in the network and
          #a store for the trains that were in the network

trains_now=[0 for x in range(network_length+1) ]
#A vector containing all the trains (as classes) which are on the network,
                            #the zero position are for the trains getting out
trains_old=set()   #Trains that were in the rail
matrix_of_trains=[[0 for x in range(network_length)] for x in range(time_steps)]
#output matrix for the csv
#here we keep track of the transition by one state to another (time is discrete)
transitions=[0 for x in range(time_steps)] #One vector per time step
starting_edges = ord_nodes[-1]
for t in range(time_steps):
transitions[t]=new_move(trains_now,nodes,ord_nodes,network_length,
                                    prob_stop_anywhere,prob_stop_station)
trains_now=refresh(trains_now,transitions[t],network_length)
trains_now, trains_old=store(trains_now,trains_old, t)

for edge in starting_edges:
if trains_now[edge]==0 and np.random.rand()<prob_inject:
train_type = np.random.randint(2)
train_delay = np.random.poisson(lambda_delay)
trains_now = new_train(edge,trains_now,t-train_delay,train_type,train_delay)
#we save the trains positions and categories at each step
for i in range(network_length): #We avoid the first entry that is the OUT one
if trains_now[i+1]==0:   #Empty node
matrix_of_trains[t][i]=0
else: #if it is a train
matrix_of_trains[t][i]=trains_now[i+1].category+1
#We add one because our category  start from zero


return trains_old,trains_now,transitions,matrix_of_trains

#Output of all the trains arrived, with starting time,
                      #ending_time, initial delay and category

def single_matrix_output(ad_matrix,output_length,prob_inject,lambda_delay,
                                            prob_stop_anywhere,prob_stop_station):

network_length = ad_matrix.shape[0]
#first we generate the nodes vector:
nodes=generate_nodes(ad_matrix,network_length)
#We order the nodes by priority for making decision, starting from the end of the
```

```
ord_nodes=ordering_nodes(ad_matrix, network_length)
#Here we initialize the trains in the network and a store for the trains that were

trains_now=[0 for x in range(network_length+1) ]
#A vector containing all the trains (as classes) which are on the network,
                           #the zero position are for the trains getting out
trains_old=set()   #Trains that were in the rail
matrix_of_trains=[[0 for x in range(4)] for x in range(time_steps)]
#output matrix for the csv
#here we keep track of the transition by one state to another (time is discrete)
transitions=[0 for x in range(time_steps)] #One vector per time step
starting_edges = ord_nodes[-1]

while len(trains_old)<output_length:
transitions[t]=new_move(trains_now,nodes,ord_nodes,network_length,
                              prob_stop_anywhere,prob_stop_station)
trains_now=refresh(trains_now,transitions[t],network_length)
trains_now, trains_old=store(trains_now,trains_old, t)

for edge in starting_edges:
if trains_now[edge]==0 and np.random.rand()<prob_inject:
train_type = np.random.randint(2)
train_delay = np.random.poisson(lambda_delay)
trains_now = new_train(edge,trains_now,t-train_delay,train_type,train_delay)

counter=0
for i in trains_old: #We save in a matrix all the caracteristic of the arrived tra
matrix_of_trains[counter]=[i.starting_time,i.ending_time,i.delay,i.category]
counter+=1

return matrix_of_trains

def which(listone,condizione):
return [index for index, item in enumerate(listone) if item == condizione]

def check(time_step,prob_inject):
lambda_pois = 1
prob_stop_station = 0.05
prob_stop_anywhere = 0.005
a,b,c,matrix=railtrack(ad_matrix,time_step,prob_inject,lambda_pois,
                                    prob_stop_anywhere,prob_stop_station)
b1=[x.ending_time-x.starting_time for x in a if x.category==1]
b2=[x.ending_time-x.starting_time for x in a if x.category==0]
return np.histogram(b1,10), np.histogram(b2,10), np.mean(b1), np.mean(b2)
```

```
def outputcsv(time_step):
a,b,c,matrix_output=railtrack(ad_matrix,17,time_step)
import csv
with open("output.csv","wb") as csvfile:
outp = csv.writer(csvfile, delimiter = ",")
for i in range(time_step):
outp.writerow(matrix_output[i])
return

'''
```

Code in Python2:

A.2 Second Appendix: Perl code for Signalling Dynamics

```
#!/usr/bin/perl - Tw

use strict;
use Math::Trig;

my $N = 10;       # number of trains
my $L = 100;      # length of line (number of signals)
my $T = 30;       # total time to integrate over
my $dt = 0.01;    # timestep

my $d = 0.1;      # train length
my $h = 0.3;      # visibility distance for next signal

my ($v0,$v1,$v2,$v3)  # desired speed for each signal aspect
    = (0.0,1.0,1.8,2.0);
my $a0 = 3;       # max acceleration
my $vscl = 0.1;   # velocity scale for acceleration


my $maxits = $T/$dt;

my (@x,@v,@sigma,$dx,$dv);
my (@s);
my @train;

for (my $n=0;$n<$N;$n++) {
    $x[$n] = 1*$n+0.99;
    $v[$n] = 0;
    $sigma[$n] = 1;
    local *FILE;
    open FILE, ">train$n.dat";
    push(@train,*FILE);
}

for (my $a=0;$a<$L;$a++) {
```

```
36      $s[$a] = 0;
37  }
38
39  open POS, ">position.dat";
40
41  for (my $i=0;$i<$maxits;$i++) {
42
43      for (my $n=0;$n<$N;$n++) {
44          if (int($x[$n])+1 - $x[$n] < $h) {
45              $sigma[$n] = $s[int($x[$n])+1];
46          }
47          $dx=$dt*$v[$n];
48          $dv=$dt*&Accn( $v[$n], $sigma[$n], $x[$n]-int($x[$n]) );
49          $x[$n] += $dx;
50          $v[$n] += $dv;
51      }
52
53      for (my $a=0;$a<$L;$a++) {
54          my $tmp = 3;
55          for (my $n=0;$n<$N;$n++) {
56              if ($x[$n]>$a && int($x[$n])-$a<$tmp) {
57                  $tmp = int($x[$n])-$a;
58              } else {
59              }
60              if ($x[$n]>$a+$d && int($x[$n]-$d)-$a<$tmp) {
61                  $tmp = int($x[$n]-$d)-$a;
62              }
63          }
64          $s[$a]=$tmp;
65      }
66
67      for (my $n=0;$n<$N;$n++) {
68          my $fh = $train[$n];
69          printf $fh "%g %g %g %g\n",
70                 $i*$dt, $x[$n], $v[$n], $sigma[$n];
71      }
72
73      printf POS "%g %g %g %g %g %g %g %g %g %g %g\n", $i*$dt, @x;
74
75  }
76
77  close POS;
78
79  for (my $n=0;$n<$N;$n++) {
80      close $train[$n];
81  }
82
83  sub Accn {
84      my ($u,$sigma,$delta)=@_;
85      return $a0*tanh((&Vdes($sigma)-$u)/$vscl);
86  }
87
88  sub Vdes {
89      my $s = @_[0];
```

```
90      my %speed = (
91          0 => $v0,
92          1 => $v1,
93          2 => $v2,
94          3 => $v3,
95          );
96      return $speed{$s};
97  }
```

A.3 Third appendix: Code in *Mosel* modelling language to be solved by *Xpress* for Mixed Integer Programming

```
```Mosel
model "esgi-1"
version 1.862

uses "mmxprs", "mmsheet", "mmsystem"

declarations
  ! large constant
  MM = 1000

  ! index sets
  EVENTS : set of integer
  TRAINS : set of integer
  STATIONS : set of string
  CLASSES = 1..2

  ! track data
  HW_MIN : array (STATIONS) of integer ! min headway time in each station
  TRAVEL_MIN : array(STATIONS, STATIONS, CLASSES) of real   ! min travel times or

  ! train data
  t_sigid : array (TRAINS) of string ! train signal id ("2M23")
  t_class : array (TRAINS) of integer ! class of train
  t_dir   : array (TRAINS) of integer ! direction of train (0 = up = to Pad, 1 = d

  ! events
  e_plan  : array (EVENTS) of real ! scheduled time of event
  e_public: array (EVENTS) of integer ! is plan publicly advertised?
  e_type  : array (EVENTS) of integer ! 0 ... arrival; 1 ... departure
  e_train : array (EVENTS) of integer ! train id of event
  e_stn   : array (EVENTS) of string ! station id of event
  e_next  : array (EVENTS) of integer ! next event on the train
```
```

```
  ! decision variables
  e_time : array (EVENTS) of mpvar ! newly scheduled times for events
  prec  : array (EVENTS,EVENTS) of mpvar   ! which event precedes which

  ! objective
  TotalDelay : linctr
end-declarations

! add headway constraints for two events
procedure add_headway(e1, e2 : integer, hwmin : real)
  ! need to know which event precedes which
  create(prec(e1,e2))
  create(prec(e2,e1))
  prec(e1,e2) is_binary
  prec(e2,e1) is_binary
  prec(e1,e2) + prec(e2,e1) = 1

  ! headway constraints
  Headway(e1,e2) := e_time(e2) - e_time(e1) >= hwmin - MM * prec(e2,e1)
  Headway(e2,e1) := e_time(e1) - e_time(e2) >= hwmin - MM * prec(e1,e2)
end-procedure

! headway constraints for two trains on a track segment
procedure add_two_headways(e1, e2 : integer)
  ! check it's departures
  if (e_type(e1) <> 1 or e_type(e2) <> 1) then
    writeln("Events ", e1, ", ", e2, " cannot be headwayed: not departures!")
    exit(1)
  end-if

  ! check stations
  stn1 := e_stn(e1)
  if (e_stn(e2) <> stn1) then
    writeln("Events ", e1, ", ", e2, " cannot be headwayed: different stations!")
    exit(1)
  end-if

  e3 := e_next(e1)
  e4 := e_next(e2)
  stn3 := e_stn(e3)
  if (e_stn(e4) <> stn3) then
    writeln("Events ", e1, ", ", e2, " cannot be headwayed: different departure li
    exit(1)
  end-if
```

```
  ! add the headway constraints
  add_headway(e1, e2, HW_MIN(stn1))
  add_headway(e3, e4, HW_MIN(stn3))

  ! preserve precedence
  Prec(e1, e2, e3, e4) := prec(e1, e2) = prec(e3, e4)
end-procedure

! generating travel/dwell time constraints
procedure add_travel(e1, e2 : integer)
  ! check train
  ! trn := e_train(e1)
  ! if (e_train(e2) <> trn) then
  !   writeln("Events ", e1, ", ", e2, " cannot be travel-separated: different tra
  !   exit(1)
  ! end-if

  TravelMin(e1, e2) := e_time(e2) - e_time(e1) >= TRAVEL_MIN(e_stn(e1),e_stn(e2),t
end-procedure

writeln("Reading in data: ", gettime)

! data init
initializations from "topology.dat"
  HW_MIN
  TRAVEL_MIN
end-initializations

initializations from "mmsheet.xlsx:events.xlsx"
  e_train as "[a2:g119](1,2)"
  e_stn as "[a2:g119](1,3)"
  e_type as "[a2:g119](1,4)"
  e_plan as "[a2:g119](1,5)"
  e_public as "[a2:g119](1,6)"
  e_next as "[a2:g119](1,7)"
end-initializations

initializations from "mmsheet.xlsx:trains.xlsx"
  t_sigid as "[a2:d13](1,2)"
  t_class as "[a2:d13](1,3)"
  t_dir   as "[a2:d13](1,4)"
end-initializations

writeln("Generating constraints: ", gettime)
```

```
! generate travel-time constraints and dwell-time constraints for each train run
forall(e in EVENTS) do
  create(e_time(e))
  if (exists(e_next(e)) and e_next(e) <> 0) then
    add_travel(e, e_next(e))
  end-if
end-do

! headway constraints
forall(e1 in EVENTS, e2 in EVENTS |
      (e_train(e1) <> e_train(e2)) and
(e_stn(e1) = e_stn(e2)) and
(e_type(e1) = 1) and
(e_type(e2) = 1) and
(t_class(e_train(e1)) = t_class(e_train(e2)))) do
  if (exists(e_next(e1)) and exists(e_next(e2))) then
    en1 := e_next(e1)
    en2 := e_next(e2)
    if (exists(e_stn(en1)) and exists(e_stn(en2))) then
      if (e_stn(en1) = e_stn(en2)) then
add_two_headways(e1, e2)
      end-if
    end-if
  end-if
end-do

! don't want to run before scheduled time
forall(e in EVENTS) do
  e_time(e) >= e_plan(e)
end-do

!!!!!!!!!!!! boundary conditions
!mock delay
e_time(37283) >= 10
e_time(45142) >= 34
!prec(37287,45107)=1
!prec(37289,45109)=1
!prec(37283,45103)=1
!prec(37285,45105)=1
!!!!!!!!!!!!

! objective
TotalDelay := sum(e in EVENTS | exists(e_plan(e))) (e_time(e) - e_plan(e))

writeln("Optimising: ", gettime)
```

```
! minimise
minimise(TotalDelay)

write("Finished: ", gettime, ": ")

case getprobstat of
  XPRS_OPT: writeln("optimal")
  XPRS_INF: writeln("infeasible")
  XPRS_UNB: writeln("unbounded")
  XPRS_UNF: writeln("unfinished")
else
  writeln("unexpected problem status!")
end-case


! results
writeln("Total delay: ", getobjval)

!! forall (e in EVENTS | exists(e_time(e)))
  !! writeln("Event ", e, ": time ", getsol(e_time(e)))

forall(t in TRAINS) do
  writeln; writeln(t_sigid(t),"\tarr\tdep")
  forall(e in EVENTS | e_train(e) = t) do
    write(e_stn(e))
    if (e_type(e) = 1) then
      writeln("\t\t", getsol(e_time(e)), "\t(+", getsol(e_time(e)) - e_plan(e), ")
    else
      writeln("\t", getsol(e_time(e)), "\t\t(+", getsol(e_time(e)) - e_plan(e), ")
    end-if
  end-do
end-do

end-model
```

A.4 Fourth appendix: Code in *R* for A Bayesian model of route status

```
# Resonate Bayesian Inference Model
# By Yuanwei Xu (yuanwei.xu@imperial.ac.uk)
```

```
# x: vector of ratings
# shape1, shape2: parameters of the Beta prior of the ratings
# shape1 = shape2  = 1 corresponds to uniform prior between 0 and 1
log_prior <- function(x, shape1, shape2)
  sum(dbeta(x, shape1, shape2, log = TRUE))


# vt: vector of relative lateness
# vr: vector of ratings
# set_j, set_s: index set of junctions and stations
# mu_j, mu_s, mu_t: the parameters associated with expoinential distributions of r
# operational service delay, for junctions, stations and terminal
log_likelihood <- function(vt, vr, set_j, set_s, mu_j, mu_s, mu_t){
  mu <- vector("numeric", length(vr))
  for(i in seq_along(vr)){
    mu[i] <- dplyr::case_when(
      i %in% set_j ~ mu_j,
      i %in% set_s ~ mu_s,
      i == length(vr) ~ mu_t
    )
  }
  lambda <- 1/vr - vr

  # Use the result here  https://people.maths.bris.ac.uk/~mb13434/sumexp.pdf
  foo <- dexp(vt,mu)/mu
  bar <- dexp(vt,lambda)/lambda
  out <- sum(log(mu * lambda * (foo/(lambda-mu) + bar/(mu-lambda))))
  out
}

# Posterior density
log_posterior <- function(vr, vt,
                          set_j, set_s, mu_j, mu_s, mu_t, shape1 = 1, shape2 = 1){
  log_prior(vr, shape1, shape2) + log_likelihood(vt, vr, set_j, set_s, mu_j, mu_s,
}

# The proposal changes one component of vr using a normal distribution with
# current value as mean and standard deviation 0.1
proposal <- function(vr, s = 0.1){
  prop <- vr
  pos <- sample(length(vr), 1)
  r <- rnorm(1, mean = vr[pos], sd = s)
  while(r<0 || r>1) # make sure r is in range
    r <- rnorm(1, mean = vr[pos], sd = s)
```

```
  prop[pos] <- r
  list(val = prop, pos = pos)
}

# Acceptance ratio of the MCMC sampler
acc_ratio <- function(vr_new, vr_curr, ...){
  log_posterior(vr_new, ...) - log_posterior(vr_curr, ...)
}


num_sections <- 10
set_j <- c(1, 3, 5, 7, 9)
set_s <- c(2, 4, 6, 8)
mu_j <- 2
mu_s <- 1
mu_t <- 0.5
vt <- c(0.5, 1, 0.5, 1, 0.5, 1, 0.5, 1, 0.5, 2)
vr_init <- rep(0.5, num_sections)

iters <- 20000
out <- matrix(nrow = iters, ncol = num_sections)
pb <- txtProgressBar(min = 0, max = iters, initial = 0, style = 3)
vr_curr <- vr_init
for(i in seq.int(iters)){
  setTxtProgressBar(pb,i)
  prop <- proposal(vr_curr)
  vr_new <- prop$val
  if(log(runif(1)) <
     acc_ratio(vr_new, vr_curr, vt, set_j, set_s, mu_j, mu_s, mu_t)){
    vr_curr <- vr_new
    out[i, ] <- vr_curr
  }
  else
    out[i, ] <- vr_curr
}

# Plot historgrams overlayed by mean
par(mfrow=c(2,5))
for(i in 1:num_sections){
  hist(out[,i], main = substitute(paste("Section ID: ", a), list(a=i)))
  abline(v = mean(out[,i]), col = "red", lwd = 2)
}

library(tidyverse)
out <- as.data.frame(out)
```

```
names(out) <- 1:num_sections
out %>%
  gather(section_id, rating, convert = TRUE) %>%
  mutate(section_id = as.factor(section_id)) %>%
  ggplot(aes(section_id, rating)) +
  geom_boxplot()

delay <- data.frame(section_id = 1:num_sections, lateness = vt)
delay %>%
  ggplot(aes(as.factor(section_id), lateness)) +
  geom_point() +
  ylab("relative lateness")
```

‘‘‘

References

[1] Coherent Frameworks for Statistical Inference serving Integrating Decision Support Systems, 2015, *Jim Q. Smith, Martine J. Barons & Manuele Leonelli*, arXiv: 1507.07394

[2] Eliciting Probabilistic Judgments for Integrating Decision Support Systems, *Martine J. Barons, Sophia K. Wright and Jim Q. Smith* in: Dias LC, Morton A, Quigley J (eds), Elicitation: The science and art of structuring judgement. Springer, New York, 2018 (Chapter 17)

[3] Decision focused inference on networked probabilistic systems: with applications to food security, *Jim Q. Smith, Martine J. Barons, Manuele Leonelli*, Conference paper JSM2015 - Section on Bayesian Statistical Science

[4] Decision with Multiple Objectives: Preferences and Value Trade-offs. 1993 *R. L. Keeney and H. Raiffa. , R. L. Keeney and H. Raiffa.* Cambridge University Press.

[5] Netica Norsys Software Corp

`https://www.norsys.com/netica.html`

[6] EFSA (2014). Guidance on expert knowledge elicitation in food and feed safety risk assessment. European Food safety Authority (EFSA) Journal 12(6), 3734.

[7] Investigate discuss estimate aggregate for structured expert judgement. *Hanea, A., M. McBride, M. Burgman, B. Wintle, F. Fidler, L. Flander, C. Twardy, B. Manning, and S. Mascaro* (2016). International Journal of Forecasting.

[8] Eliciting probabilities from experts in complex technical problems. *Keeney, R. and D. von Winterfeldt* (1991).IEEE Transactions on Engineering Management 38, 191-201.

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

[10] Rescheduling models for railway traffic management in large-scale networks *Pavle Kecman, Francesco Corman, Andrea D'Ariano, Rob M. P. Goverde.* Public Transport September 2013, Volume 5, Issue 1–2, pp 95–123

[11] Modelling train delays with q-exponential functions *Keith Briggs, Christian Beck.* Physica A 378 (2007) 498–504

[12] Analyzing passenger train arrival delays with support vector regression. 2015. *N. Markovic, S. Milinkovic, K. S. Tikhonov and P. Schonfeld.* Transport. Research Part C: Emerging Tech. 56, 251-262

[13] Delay prediction system for large-scale railway networks based on big data analytics. 2017. *L. Oneto, E. Fumeo, G. Clerico, R. Canepa, F. Papa, C. Dambra, N. Mazzino and D. Anguita* Advances in Intelligent Systems and Computing. 529, 139-150

[14] Railway passenger train delay prediction via neural network model. 2013. *M. Yaghini, M. M. Khoshraftar and M. Seyedabadi* J. Adv. Transport. 47(3), 355-368

[15] Burkolter, Dan Max. Capacity of railways in station areas using petri nets. Diss. 2005, ETH Zürich GiuaGiua, Alessandro, and Carla Seatzu. "Modeling and supervisory control of railway networks using Petri nets." IEEE Transactions on automation science and engineering 5.3 (2008): 431-445; Modelling of a railway network including sensors and semaphores

[16] Banik, Mandira, and Sudeep Ghosh. "Railway Network Modelling Using Petri Nets." International Journal of Science, Engineering and Computer Technology 3.7 (2013): 249-255; Usage of Petri nets to provide a control logic to avoid collisions

[17] Ricci, S., and A. Tieri. "A petri nets based decision support tool for railway traffic conflicts forecasting and resolution." WIT Transactions on the Built Environment 103 (2008): 483-492; The proposed model allows to study railway traffic regularity and can be used as a decision support tool and was tested on the line Roma-Formia.

[18] Venkatarangan, M. J., Anthony Man-Cho So, and Yam Yeung. "Modelling, Simulation and Optimisation of Train Traffic with Passenger Movement." International Journal of Simulation–Systems, Science & Technology 16.3 (2015);

Modelling train arrivals and departures as well as passenger movements with the aim to optimise train schedules

[19] Ren, Xin, and MengChu Zhou. "Tactical scheduling of rail operations: a Petri net approach." Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on. Vol. 4. IEEE, 1995; Presentation of Petri net models for tactical scheduling of railroad operations

[20] Sakaguchi, T., and N. Tomii. "A train traffic model based on coloured Petri Nets and its application to a train schedule planning system." WIT Transactions on The Built Environment 20 (1970); Introduction of a train traffic model based on coloured Petri nets allowing simulation to make a train schedule

[21] Bartkevicius, S., et al. "Train traffic simulation with coloured petri nets and schedule optimisation." Elektronika ir elektrotechnika 59.3 (2005); Coloured Petri nets used to describe parameters of tracks and stations, with tokens containing all train parameters for simulation

[22] Caetano, L. F., and P. F. Teixeira. "Stochastic Train Delay Simulation using Petri Nets." Civil-Comp Press, Proceedings of the Second International Conference on Railway Technology: Research, Development and Maintenance, J. Pombo, (Editor), Civil-Comp Press, Stirlingshire, Scotland (2014)

`https://www.researchgate.net/profile/Luis_Caetano/publication/269198580_Stocha`

[23] Caetano and Teixeira (2014) similarly model delay propagation probabilistically by considering stochastic Petri nets.

[24] Daamen, Winnie, Rob MP Goverde, and Ingo A. Hansen. "Non-discriminatory automatic registration of knock-on train delays." Networks and Spatial Economics 9.1 (2009): 47-61; Construction of a prototype tool for identification of conflicts and estimation of knock-on delay, tested, with success, on the Dutch railway network.

[25] A fuzzy Petri net model to estimate train delays. 2013. *S. Milinkovic, M. Markovic, S. Veskovic, M. Ivic and N. Pavlovic*. Simulation Modelling Practice and Theory. 33, 144-157

[26] Stochastic Train Delay Simulation using Petri nets. In: Proceedings of the Second International Conference on Railway Technology: Research, Development and Maintenance. J. Pombo (Ed.) *L. F. Caetano and P. F. Teixeira* Civil-Comp Press, Stirlingshire, Scotland.

[27] Giua, Alessandro, and Carla Seatzu. "Modeling and supervisory control of railway networks using Petri nets." IEEE Transactions on automation science and engineering 5.3 (2008): 431-445; Modelling of a railway network including sensors and semaphores