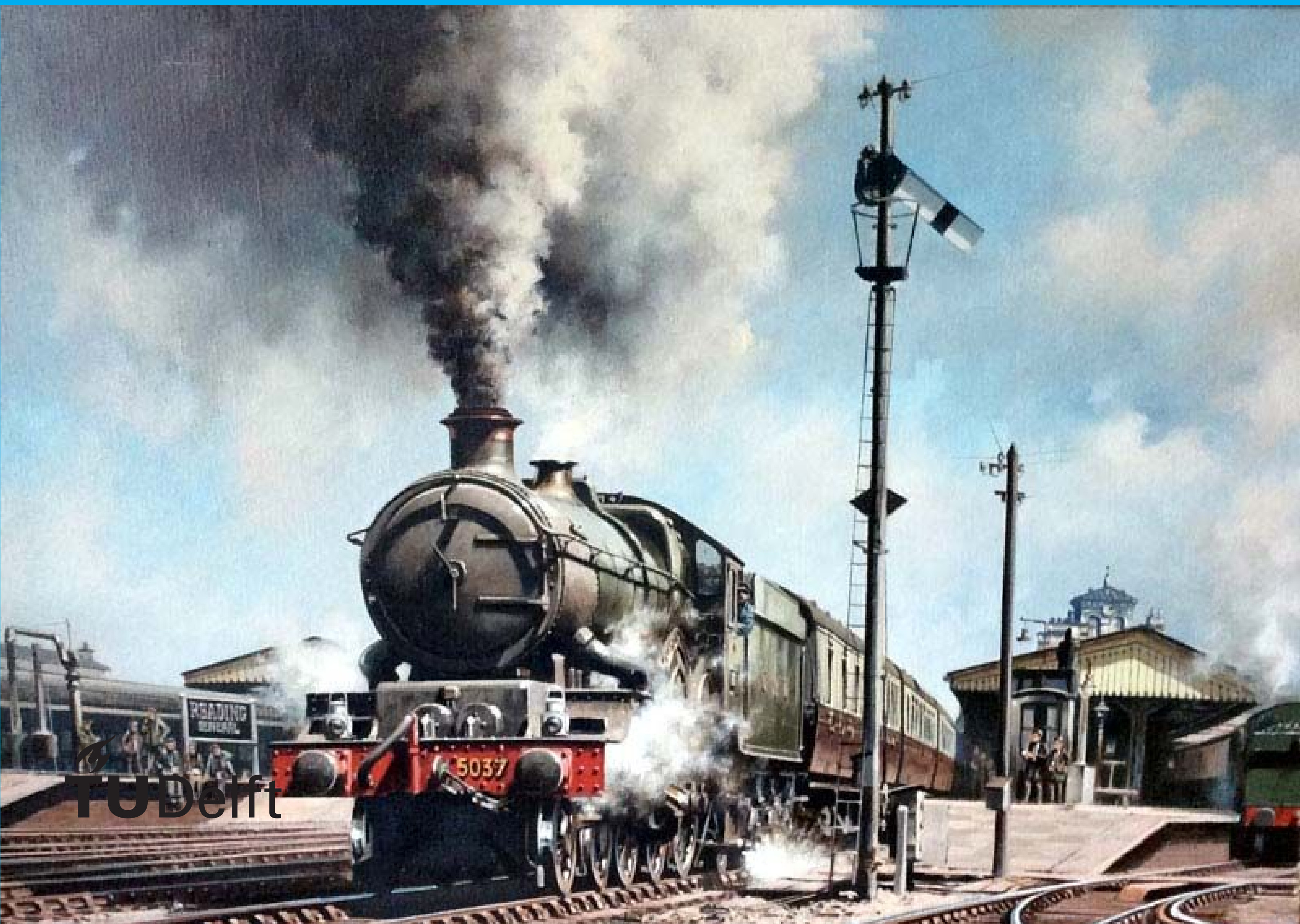
Max-Plus Extensions

A Study of Train Delays

Gideon Vissers

Bachelor Thesis
Applied Mathematics

TU Delft



Max-Plus Extensions

A Study of Train Delays

by

Gideon Vissers

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday June 24, 2022 at 13:00.

Student number:	4998081	
Project duration:	April 18, 2022 – June 24, 2022	
Thesis committee:	Dr. J.W. van der Woude,	TU Delft, supervisor
	Prof. Dr. Ir. M.B. Van Gijzen,	TU Delft

TU Delft

Summary

In this report, we research a model for trains called the max-plus model. We start by introducing the model and the underlying mathematics, after which we set out to extend the model so that we can influence the network ourselves. The goal we formulate in the report is the resolution of delays occurring in train networks. To achieve this goal, we design delay resolution methods. We first verify that these methods work under some conditions, after which we compare and evaluate them. The comparison and evaluation is done using a simulation of a train network where delays are added at random. In this simulation, the delay resolution methods are applied to determine the optimal method. Though the report is written with the narrative of resolving delays, an added objective is to further study applications and interactions of max-plus algebra, more specifically max-plus models and their extensions, as well as form an intuition of how problems in max-plus algebra can be solved computationally.

In chapters 2 and 3, we introduce the max-plus model and an important extension. After this, we formulate the delay problem: What is the most optimal way to resolve delays. Based on this formulation, we design delay resolution methods. The most important method is the p-greedy delay resolution methods. This method seeks to steer the network to the most favourable state in the next p time steps. An interpretation of what a favourable state constitutes is given in chapter 5. In this chapter, we also introduce obstacles to the network that inhibit our ability to resolve delays. Chapter 6 is dedicated to the aforementioned simulation. In this chapter we discuss how the simulation was constructed and apply the delay resolution methods to a day worth of trains. Here, we were able to determine that the p-greedy delay resolution methods performs best for higher values of p on average. This corresponds to the algorithm thinking further ahead when resolving delays. Though this result is true on average, in specific cases, other values for p are better.

In order to achieve the above results, the switching max-plus extension was formalised and the multi-switching extensions was designed to make the model more realistic. The delay resolution methods are based on an intuitive approach, as opposed to obscure theoretical or statistical methods. The intuition behind these methods can thus be applied to a larger class of computational optimisation or calamity resolution problems. In addition to the latter, the intuitive nature of the methods also allow humans to intervene in max-plus modellable networks so solve issues. This is especially interesting in networks where complicated black-box algorithms are used to regulate the network, such as train networks; if the powerful black-box systems fail or malfunction, networks can be kept operational through human intervention.

The biggest accomplishments of this report are the formalisation of the max-plus extensions, the formulation of the delay problem and delay resolution methods and the design and implementation of the train simulation. It is hoped that these achievements can lay the foundation for further max-plus or logistics related research.

Contents

1 Introduction **1**

2 Max-Plus Algebra **2**
2.1 Trains and Transfers 2
2.2 The Max-Plus Algebraic Structure 3
2.2.1 Properties of Max-Plus Algebra 4
2.2.2 Max-Plus Matrices and Vectors 4
2.2.3 Eigenvalues and Eigenvectors 5
2.3 Generalising the Max-Plus Model 6
2.4 Finding eigenvalues: The Power Method 10

3 Switching Max-Plus **11**
3.1 Example of a Switching System 11
3.2 Definitions 12
3.3 Delayed Trains 13
3.4 Timetable Improvements 16

4 Delay Problems **17**
4.1 Solving the Problem: Every Possibility 17
4.1.1 The Combinatorial Method 17
4.1.2 The Minimal Solution 20
4.1.3 Computational Restrictions 21
4.2 Sub-Optimal Methods 21
4.2.1 The Greedy Delay Resolution Method 22
4.2.2 The Composite Greedy Delay Resolution Method 25
4.2.3 Time Complexity 26
4.3 Calamity management: Decoupling 27
4.3.1 Severe Delays 27
4.3.2 Decoupling Conditions 29
4.3.3 Structural Decoupling 29
4.3.4 Early Departures 30
4.4 Network Design 30

5 Multi-Switching Max-Plus **33**
5.1 Freight Train Obstruction 33
5.2 Departure Score 36
5.3 Modelling Delays 38
5.3.1 Delayed Classes 38
5.3.2 Anterior and Posterior Indexing 39
5.3.3 Delays in Simulations 40
5.3.4 Implementation of Adjacency Classes 41
5.4 The Scoring Problem 42

6 Simulating Train Networks **45**
6.1 The Simulated Network 45
6.2 Implementing Dynamic Delays 47
6.2.1 Modelling Random Delays 47
6.2.2 Probabilistic Parameters 49
6.2.3 Example Delayed Sequences 50

6.3 Applying Delay Resolution . . . 51
6.4 Speed Up and Decoupling Restrictions . . . 53
6.5 Resolution Evaluation Criteria . . . 54
6.6 Results . . . 55

7 Conclusion **57**
7.1 Max-Plus Modelling Results . . . 57
7.2 Delay Resolution Results . . . 57
7.3 Network Simulation Results . . . 58
7.4 Overall Results . . . 58
7.5 Further Research . . . 58

A Residual Proofs and Derivations **60**
A.1 Max-Plus Algebra . . . 60
A.2 Switching Max-Plus . . . 61
A.3 Delay Problems . . . 62

B Decoupling **63**
B.1 Severe Delays . . . 63
B.2 Decoupling Criteria . . . 64
B.3 Rush Hour Modelling . . . 65
B.3.1 Modelling Rush Hour . . . 65
B.3.2 Transitioning to Rush Hour . . . 66
B.3.3 Method Parameters . . . 67

C Systems and Control for Max-Plus Algebra **68**
C.1 Max-Plus, Systems and Slow-Downs . . . 68
C.2 Slowing Down instead of Speeding Up . . . 69
C.3 State Controller . . . 69
C.4 Delay Resolution and Control . . . 70

D Python Code **71**
D.1 Classes and Functions . . . 71
D.1.1 Classes . . . 71
D.1.2 Functions . . . 83
D.2 Examples . . . 85
D.3 Simulation . . . 92
D.3.1 Simulation Initialisation . . . 92
D.3.2 Simulation . . . 93

1 Introduction

Workflow has become a cornerstone of developed civilisation. The ability to efficiently fulfil tasks is key to all manner of developments in all sectors of society. Now more than ever, people can employ their skills at those places where they are most desired, due to our ability to transport ourselves. This is illustrated by the fact that there is a strong correlation between transport infrastructure and regional economic growth (Hong et al., 2011). As such, it is paramount that the integrity of transport networks are safeguarded by preventing decreases in their logistic capacity. Despite our best efforts, many people are still regularly plagued by logistic failures such as train delays, grounded aeroplanes and traffic jams. In this report, we will analyse such transport networks and formalise some of the logistic problems that occur within them, in the hopes of decreasing the negative impact they have on our ability to commute and ultimately on our ability to contribute to society.

Trains are one of the major transport networks used today. In just one day, Germany's railway network transports nearly 12 million passengers ("Facts and figures 2016", 2016). With this magnitude of commuters, even small delays can lead to massive collective time loss. Furthermore, a major issue with delays, is that one delay can cause several other delays. If delays are left to propagate like this, a delay at the beginning of the day can render a train network useless for the remainder, causing many passengers to be late to important meetings or unable to fulfil their societal duties. Because of this, resolving delays as soon as possible and with little propagation is very important, but often far from simple. Systems are in place to make sure that delays are resolved and train commutes match their timetables as closely as possible, but due to the seeming complexity of these system, their failure can have catastrophic results, such as a total halt in all train traffic (Middelkoop, 2022). If such system failures occur, having more intuitive systems in place allowing for people to monitor the train traffic and intervene when necessary, allows for large parts of the system to still be functional, preventing a nation-wide catastrophe.

In this report, we will explore one such intuitive systems, namely so-called max-plus models. Max-plus systems are systems that can describe processes where new tasks can only be started once old tasks are completed, yielding the 'max' component. The model tracks the start time of new tasks provided the amount of time each task requires, yielding the 'plus' component. In the context of trains, these tasks can be interpreted as the commute a train makes from one station to the next. In this report, we will be modelling trains using max-plus models, disjoint from any existing timetable systems. The previously discussed obstacle and the main subject of this report is the delay propagation in train networks. We can summarise the main problem we seek to solve as: **'What is the best way to solve a delay in a train network'**. For the time being, this formulation is quite vague. Throughout the report, we will provide mathematical definitions and insights that allow us to reformulate this problem in a more refined manner.

2

Max-Plus Algebra

In this chapter, we will briefly discuss max-plus algebra. We will use an example to derive a max-plus model for modelling time tables of public transport, in our case trains. We will then define what max-plus algebra entails and state some important properties. We will finish off by introducing the concept of eigenvalues and eigenvectors in max-plus, establish how to interpret them and finally look at how to calculate them. The contents of this chapter are heavily based on the book 'Max Plus at Work' (Heidergott et al., 2006).

2.1. Trains and Transfers

To illustrate all mathematics in this section, we consider a train network, as seen in figure 2.1.

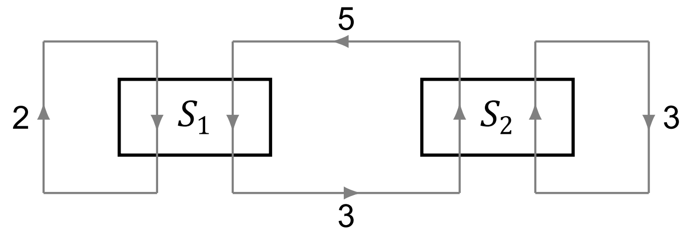


Figure 2.1: The subject train network

This train network can also be found in the book Max Plus at Work (Heidergott et al., 2006). In this network, we observe 2 train stations, S_1 and S_2. The lines, also called arcs between the train stations correspond to train tracks. The arrows on these rails show in which direction the train drives and the numbers show how long it takes for a train to traverse that set of tracks. We now place one train on each of the tracks leaving the stations, this means we place a total of 4 trains. We allow these trains to drive from station to station, according to the following set of rules:

- A train will drive from one station to the next in the time indicated on that arcs. Upon arrival, the train can depart immediately if all other criteria are met.
- All trains at a station can depart only if all trains arriving at the station have arrived, thus allowing passengers to transfer from one train to another. Once all trains have arrived at a station, they will all immediately depart.

Now that we have established the rules of the model, we give an example to give some more insight.

Example 1 *We place the 4 trains in the stations and allow them to all leave immediately, so their departure time is 0. After 2 time units the left-most train arrives at station S_1, but the train travelling from S_2 to S_1 has not arrived yet, so the train can not leave. At time 3, both trains have arrived at station*

S_2 and so both trains can leave. At time 5, the second train arrives at station S_1, so both trains can leave. This gives the following departure times:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 3 \end{pmatrix}.$$

The vector $(0,0)^T$ shows the first departure time and $(5,3)^T$ shows the second departure time. We can continue this train of thought to generate the following sequence of departure times:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 3 \end{pmatrix} \to \begin{pmatrix} 8 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 11 \end{pmatrix} \to \begin{pmatrix} 16 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 19 \end{pmatrix}.$$

We see here that every second time step, the departure times are a multiple of 8. In addition to this, we see that consecutive departure times are not evenly spaced. When looking at station S_1 for example, we see that the difference between consecutive departure times alternates between 5 and 3. We call sequences that do have evenly spaced departure times 'regular sequences'. The problem we aim to solve with this model is that we want to determine which regular sequence of departure times is optimal, i.e. **which sequence of departure times has the property of having the smallest constant between any 2 consecutive departure times.** In this example, this sequence would be the following:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 20 \end{pmatrix}. \tag{2.1}$$

The constant between any 2 consecutive departure times is 4, this can not become smaller as the average time to traverse the inner loop in figure 2.1 is 4 (8 time units in 2 time steps), which matches this constant.

Using the above example, we will now derive a mathematical model for this problem. To do this, we first introduce some notation. We will henceforth call the vectors with the departure times the states of the system. These states change with every time step, so the state of system at time step k is denoted $\mathbf{x}(k)$, with the first departure time being $\mathbf{x}(0)$. This means an arbitrary sequence of departure times (henceforth called departure sequences) in this example looks as follows:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} \to \begin{pmatrix} x_1(1) \\ x_2(1) \end{pmatrix} \to \begin{pmatrix} x_1(2) \\ x_2(2) \end{pmatrix} \dots \to \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} \to \dots$$

We write the vector with the stations before the colon to clearly signify which departure time corresponds to which station. We now consider station S_1. Following the rules, The arrival times of all trains is the departure time at their previous station added to the travel time, so for the train in the left-most cycle, the arrival at station S_1 after a departure at time step k is $x_1(k) + 2$ and for the train travelling from S_2 to S_1, the arrival time is $x_2(k) + 5$. Since the trains have to wait until both trains have arrived, the new departure time is equal to the maximum of the two arrivals:

$$x_1(k+1) = \max(x_1(k) + 2, x_2(k) + 5).$$

The same intuition can be applied to station S_2, so the full model for this example becomes:

$$\begin{cases} x_1(k+1) = \max(x_1(k) + 2, x_2(k) + 5) \\ x_2(k+1) = \max(x_1(k) + 3, x_2(k) + 3) \end{cases}$$

This model is called a max-plus model due to the fact that it entirely consists of taking maxima and additions. We can easily calculate that all the departure sequences stated in this example match the above recurrence relation. From this recurrence relation we can also see that a departure sequence is entirely decided by its first entry, $\mathbf{x}(0)$.

2.2. The Max-Plus Algebraic Structure

In order to study the discussed max-plus model and related models, we create an algebraic structure using the key operators max and plus.

Definition 1 *Let* $\mathbb{R}_{max} = \mathbb{R} \cup \{\varepsilon\}$*. Define the operators* $\oplus$ *and* $\otimes$ *such that* $\forall a, b \in \mathbb{R}_{max}$*:*

$$a \oplus b = \max(a, b) \qquad a \oplus \varepsilon = a$$
$$a \otimes b = a + b \qquad a \otimes \varepsilon = \varepsilon$$

We call the algebraic structure $(\mathbb{R}_{max}, \oplus, \otimes)$ *the max-plus algebra.*

We will occasionally refer to $(\mathbb{R}, +, \times)$, the standard arithmatic algebraic structure, as 'plus-times algebra' for convenience. Note that in the above definition ε acts as $-\infty$. Indeed, $\max(a, -\infty) = a$ and $a + (-\infty) = -\infty$. We will also often denote $0 \in \mathbb{R}_{max}$ as e, which is the identity element for the max-plus multiplication. Armed with this definition, we can rewrite the max-plus model applied to the example as:

$$\begin{cases} x_1(k+1) = x_1(k) \otimes 2 \oplus x_2(k) \otimes 5 \\ x_2(k+1) = x_1(k) \otimes 3 \oplus x_2(k) \otimes 3 \end{cases}$$

Where we use the standard order of operations by calculating the max-plus multiplications first. We can compound max-plus multiplications using powers, denoted $a^{\otimes k}$, which is equal to the max-plus product $a \otimes a \otimes ... \otimes a$, where a is repeated k times. Note that $a^{\otimes k} = k \times a$ in plus-times algebra.

2.2.1. Properties of Max-Plus Algebra

Now that we have established a definition for max-plus algebra, we determine some properties of the structure. We start by summing up the algebraic properties of max-plus algebra:

- Both operations are closed.
- Both operation are associative.
- Both operations are commutative.
- $\otimes$ is distributive with regards to $\oplus$.
- There is a zero element, or additive identity: ε.
- There is a unit element, or multiplicative identity: $e = 0$.
- The zero is absorbing for $\otimes$: $a \otimes \varepsilon = \varepsilon$.
- $\oplus$ is idempotent: $a \oplus a = a$

These properties imply that max-plus algebra is a commutative, idempotent semiring. We now briefly discuss inverses to give us insight in how max-plus algebraic problems can be solved. It is easy to see that a multiplicative inverse exists for any element except ε, namely $a^{\otimes(-1)} = -a$. The additive inverse only exists for the element ε. Because of this, not all algebraic operations we are familiar with can be performed. Consider for example the following equation:

$$5 \otimes x \oplus 2 = 1.$$

When working in $(\mathbb{R}, +, \times)$, we can simply subtract 2 from both sides and continue solving the equation, but in max-plus algebra, subtraction is not allowed as there is no additive inverse. This means certain linear equations can not be solved. The above equation is an example of this fact.

2.2.2. Max-Plus Matrices and Vectors

We saw before that the states in a max-plus model can be vectors. Because of this, we introduce vectors and matrices in the max-plus sense to further simplify notation for the max-plus model. This notation will eventually even allow us to consider eigenvalues and eigenvectors.

Definition 2 *Let $\mathbb{R}^{n\times m}_{max}$ be such that $\forall A \in \mathbb{R}^{n\times m}_{max}$, A is an $n \times m$ matrix with components $a_{ij} \in \mathbb{R}_{max}$, we call these matrices max-plus matrices. We write $\mathbb{R}^{n\times 1}_{max} = \mathbb{R}^{n}_{max}$ and call its elements max-plus vectors. We define max-plus matrix addition $\oplus$ and max-plus scalar multiplication $\otimes$ as follows for $A \in \mathbb{R}^{n\times m}_{max}, B \in \mathbb{R}^{k\times l}_{max}, c \in \mathbb{R}_{max}$:*

$$[A \oplus B]_{ij} = a_{ij} \oplus b_{ij} \qquad\qquad [c \otimes A]_{ij} = c \otimes a_{ij}$$

$$[A \otimes B]_{ij} = \bigoplus_{k} (a_{ik} \otimes b_{kj})$$

For n, m, k, l such that these expressions make sense.

This definition matches the traditional definitions for matrices, vectors, matrix addition and scalar multiplication with the all sets and operations exchanged for their max-plus counterpart. We note that as usual, matrix multiplication is not commutative. We interpret powers for matrices the same as for numbers.

We now once again return to the model established in section 2.1:

$$\begin{cases} x_1(k+1) = x_1(k) \otimes 2 \oplus x_2(k) \otimes 5 \\ x_2(k+1) = x_1(k) \otimes 3 \oplus x_2(k) \otimes 3 \end{cases}$$

We observe that this recurrence relation is equivalent to the following:

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k)$$

where

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}$$

From this recurrence relation we can easily see that the state at the k'th time step can be determine using the following formula:

$$\mathbf{x}(k) = A^{\otimes k} \otimes \mathbf{x}(0).$$

Once again showing that an entire departure sequence is determined solely by the first departure.

2.2.3. Eigenvalues and Eigenvectors

Since we have defined matrices and vectors in the max-plus sense along with their operations, we can consider the equation

$$A \otimes \mathbf{v} = \mu \otimes \mathbf{v}.$$

Which defines eigenvalues and eigenvectors for max-plus matrices. In this equation, μ is an eigenvalue of A with corresponding eigenvector $\mathbf{v}$. We notice that the right hand side of the equation simply gives a translation of $\mathbf{v}$ of magnitude μ of each component:

$$[\mu \otimes \mathbf{v}]_i = \mu + [\mathbf{v}]_i.$$

In other words, the multiplication of $\mathbf{v}$ by A causes all elements of $\mathbf{v}$ to be increased by μ. This property of eigenvalues previously appeared in the problem we stated in section 2.1, where we wanted to find a constant such that all differences of the departure times in consecutive states are equal to that constant. Since the problem we stated involved finding the smallest such constant, we can rephrase the problem as **find the smallest, finite eigenvalue of A and a corresponding eigenvector.** Since we established that a departure sequence is entirely determined by its first entry, finding such an eigenvalue-eigenvector pair will ensure that we generate an optimal regular departure sequence.

We now show some properties of eigenvalues and eigenvectors to more easily find and use them. First of all, we note that due to commutativity of max-plus scalar multiplication, exponents of the matrix carries over to the eigenvalue:

$$A^{\otimes k} \otimes \mathbf{v} = \mu^{\otimes k} \otimes \mathbf{v}$$

where μ is an eigenvalue of A with corresponding eigenvector $\mathbf{v}$. We now also state an important property of eigenvectors, which is paralleled by eigenvectors in the plus-times structure, namely that a multiple of an eigenvector is also an eigenvector belonging to the same eigenvalue:

$$A \otimes (c \otimes \mathbf{v}) = \mu \otimes (c \otimes \mathbf{v})$$

Where $(\mu, \mathbf{v})$ is an eigenvalue-eigenvector pair and $c \in \mathbb{R}_{max}$ is a constant. So for any eigenvector of a matrix, we can add a constant to each of its components and it will still be an eigenvector. This is a useful result as it allows us to simplify some calculations by setting the entries of an eigenvector to be as small as possible without losing its eigenvector properties. We will illustrate this last property with an example.

Example 2 *Consider the max-plus matrix in our example:*

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}.$$

In section 2.1, we found that this matrix has the eigenvalue $\mu = 4$ *with a corresponding eigenvector* $\mathbf{v} = (1, 0)^T$. *We now consider the following state:*

$$\mathbf{x} = \begin{pmatrix} 2253 \\ 2252 \end{pmatrix}.$$

Since we can add a constant to each component without losing the eigenvector properties, we choose the constant -2252 *to add to both components. This gives the state:*

$$\mathbf{x}' = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

from which we can clearly see that since $\mathbf{x}'$ *is an eigenvector of* A, $\mathbf{x}$ *is one as well, meaning we can determine* $A \otimes \mathbf{x}$ *without any difficult calculations. This also implies that the initial state* $(2253, 2252)^T$ *will generate the same departure sequence as the initial state* $(1, 0)^T$, *translated by the constant* 2252.

2.3. Generalising the Max-Plus Model

In this section we will provide a more general way of looking at the max-plus model. We start off by introducing a way to more concisely visualise the train network using weighted, directed graph. Indeed, when looking at figure 2.1, we can see the stations as nodes and the railway connections as edges (also called arcs) with a direction and a weight. Using this observation, we can visualise the example train network as the following graph:

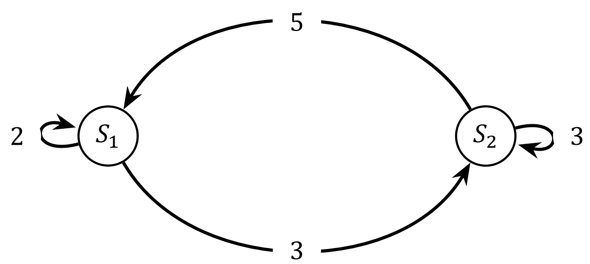


Figure 2.2: The communication graph of the example train network

We now see that the matrix A we constructed for this network

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}$$

Is the adjacency matrix of this graph. As it turns out, this observation can be used to formulate the max-plus model of any train network characterised by a directed, weighted graph.

Definition 3 *Let $G = (V, E)$ be a graph on n vertices where each vertex is labelled, so $V = \{p_i, i \in \underline{n}\}$ is the set of vertices and $E = \{e \in \binom{V}{2}\}$ is the set of edges (since G is directed, edges are ordered pairs). Let $w : E \to \mathbb{R}$ be the function mapping each edge to its weight. Let $A \in \mathbb{R}_{max}^{n\times n}$ such that:*

$$[A]_{ij} = \begin{cases} w(e), & e = (p_j, p_i) \in E \\ \varepsilon, & e = (p_j, p_i) \notin E \end{cases}$$

Then we call $A = A(G)$ the adjacency matrix of G and $G = G(A)$ the communication graph of A. $V = V(G)$ and $E = E(G)$ are called the vertex and edge sets of G respectively.

Since we will be using directed, weighted graphs a lot, we will simply call them graphs unless otherwise specified.

Method 1 *Given a graph G, the max-plus model of G can be formulated with the recurrence relation*

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k)$$

With some initial departure $\mathbf{x}(0)$, where A is the adjacency matrix of G.

In definition 3, we see that whenever there is no edge from a vertex p_i to p_j, their corresponding matrix entry is set to ε. This is because ε will not interfere with taking the maximum in the matrix multiplication, due to being the additive identity. In plus-times matrices, such entries would be set to 0, as it is the plus-times additive identity. Now that we have established the method for formulating our model, we will apply it to a slightly more intricate example to demonstrate its effectiveness.

Example 3 *Consider the following communication graph:*

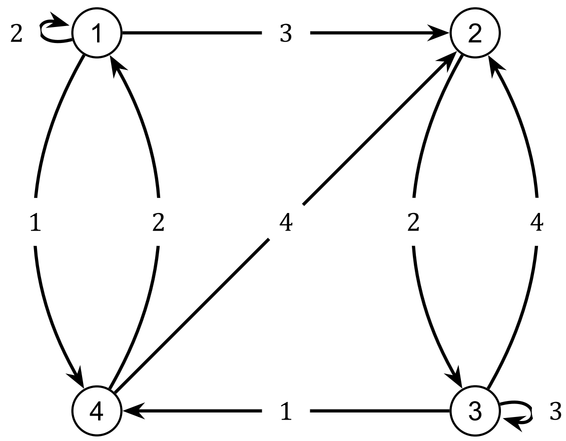


Figure 2.3: An example communication graph with labelled vertices

Using our method, we can swiftly determine the adjacency matrix:

$$A = \begin{pmatrix} 2 & \varepsilon & \varepsilon & 2 \\ 3 & \varepsilon & 4 & 4 \\ \varepsilon & 2 & 3 & \varepsilon \\ 1 & \varepsilon & 1 & \varepsilon \end{pmatrix}$$

Where we see that despite having a lot more arcs than our previous example, the increase in vertices causes the adjacency matrix to have many ε-elements. The departure sequence with initial departure $\mathbf{x}(0) = (0,0,0,0)^T$ can now easily be determined to be the following:

$$\begin{pmatrix}1\\2\\3\\4\end{pmatrix} : \begin{pmatrix}0\\0\\0\\0\end{pmatrix} \to \begin{pmatrix}2\\4\\3\\1\end{pmatrix} \to \begin{pmatrix}4\\7\\6\\4\end{pmatrix} \to \begin{pmatrix}6\\10\\9\\7\end{pmatrix} \to \ldots$$

Now that we have generalised the max-plus model to be applicable to every directed, weighted graphs, we give some more definitions surrounding the model to make discussing properties of the model easier. We start by giving some definitions surrounding departure sequences.

Definition 4 *Let $A \in \mathbb{R}^{n\times n}_{max}, \mathbf{x}_0 \in \mathbb{R}^n_{max}$. We call $\mathcal{M} = \mathcal{M}(A, \mathbf{x}_0)$ the max-plus model of A with initial departure $\mathbf{x}_0$ when*

$$\mathcal{M} : \begin{cases} \mathbf{x}(k+1) & = A \otimes \mathbf{x}(k) \\ \mathbf{x}(0) & = \mathbf{x}_0 \end{cases}$$

We call the sequence

$$\mathbf{S} : \mathbf{x}(0) \to \mathbf{x}(1) \to \mathbf{x}(2) \to \ldots \to \mathbf{x}(k) \to \ldots$$

the departure sequence of $\mathcal{M}$. Where $\mathbf{x}(0) = \mathbf{x}_0$, $\mathbf{x}(k) = A^{\otimes k} \otimes \mathbf{x}(0)$ and **S** *denotes the ordering of the nodes of the communication graph $G(A)$ of A. We say that $\mathbf{x}_0$ induces the departure sequence of $\mathcal{M}$.*

Definition 5 *Let $(\mathbf{x}(k))_{k\geq 0}$ be the departure sequence of $\mathcal{M}(A, \mathbf{x}(0))$.*
We call the sequence $(d(\mathbf{x}(k)))_{k\geq 0} = (\mathbf{x}(k+1) - \mathbf{x}(k))_{k\geq 0}$ the commute sequence of $\mathcal{M}$. Note that the departure sequence of a max-plus model is the sequence of the partial sums of the commute sequence plus the initial departure.

Definition 6 *Let $(\mathbf{x}(k))_{k\geq 0}$ be the departure sequence of $\mathcal{M}(A, \mathbf{x}(0))$.*
We call $\lfloor \mathbf{x}(k) \rfloor = \mathbf{x}(k) - ||\mathbf{x}(k)||_{min}$ the base of $\mathbf{x}(k)$ and $(\lfloor \mathbf{x}(k) \rfloor)_{k\geq 0}$ the base sequence of $\mathcal{M}$. We call any two departure states with the same base, translations of one another with the magnitude of the translation equal to the difference of their components.

By this above definition, $\mathbf{x} \otimes c$ is a translation of $\mathbf{x}$ with magnitude c.

Definition 7 *Let $(\mathbf{x}(k))_{k\geq 0}$ be the departure sequence of $\mathcal{M}(A, \mathbf{x}(0))$.*
If $\exists c, N \in \mathbb{N} : \forall k \geq N : \lfloor \mathbf{x}(k+nc) \rfloor = \lfloor \mathbf{x}(k) \rfloor$ for each $n \in \mathbb{N}$, then we call $(\mathbf{x}(k))_{k\geq 0}$ a periodic regime with its period being the smallest such c and its onset the smallest such N. We then have that $\mathbf{x}(k+nc) = \mathbf{x}(k) \otimes m$ for some m, we call $\frac{m}{c}$ the average commute time of the regime.

We now also state some properties of the commute and base sequence.

Theorem 1 *Let $(\mathbf{x}(k))_{k\geq 0}$ be the departure sequence of $\mathcal{M}(A, \mathbf{x}(0))$.*
The commute sequence and base sequence are not changed by translation: $\forall c \in \mathbb{R}$:

$$(d(\mathbf{x}(k)))_{k\geq 0} = (d(\mathbf{x}(k) \otimes c))_{k\geq 0}$$
$$(\lfloor \mathbf{x}(k) \rfloor)_{k\geq 0} = (\lfloor \mathbf{x}(k) \otimes c \rfloor)_{k\geq 0}$$

This means the behaviour of a departure sequence is characterised by the base of its initial departure.

We illustrate these definitions with an example.

Example 4 *Consider the max-plus model $\mathcal{M}(A, \mathbf{x}_0)$, where*

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \qquad \mathbf{x}_0 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}.$$

The departure sequence of $\mathcal{M}$ is then

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 2 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 5 \end{pmatrix} \to \begin{pmatrix} 10 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 13 \end{pmatrix} \to \begin{pmatrix} 18 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 21 \end{pmatrix} \to \dots$$

with commute sequence and base sequence respectively:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 3 \\ 5 \end{pmatrix} \to \begin{pmatrix} 5 \\ 3 \end{pmatrix} \to \begin{pmatrix} 3 \\ 5 \end{pmatrix} \to \begin{pmatrix} 5 \\ 3 \end{pmatrix} \to \begin{pmatrix} 3 \\ 5 \end{pmatrix} \to \begin{pmatrix} 5 \\ 3 \end{pmatrix} \to \dots$$
$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 2 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \dots$$

We see that the base sequence is 2-periodic, so the departure sequence is a periodic regime with period 2 and onset 0. We see that $\forall k, n \in \mathbb{N} : \mathbf{x}(k \otimes 2n) = \mathbf{x}(k) \otimes 8n$, so the average commute time of the regime is 4. This matches the state average (average of the components of the state) of the entries in the commute sequence.

Note that the onset of a periodic regime is not always equal to 0. Consider the max-plus model $\mathcal{M}(B, \mathbf{x}_0)$, where

$$B = \begin{pmatrix} 7 & 7 \\ 5 & 8 \end{pmatrix} \qquad \mathbf{x}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Then the departure sequence and base sequence are respectively

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 8 \\ 8 \end{pmatrix} \to \begin{pmatrix} 15 \\ 16 \end{pmatrix} \to \begin{pmatrix} 23 \\ 24 \end{pmatrix} \to \begin{pmatrix} 31 \\ 32 \end{pmatrix} \to \begin{pmatrix} 39 \\ 40 \end{pmatrix} \to \dots$$
$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \dots$$

So the departure sequence is a periodic regime with period 1 and onset 2.

We also note that if $\mathbf{x}_0$ is an eigenvector of A with corresponding eigenvalue μ, then the departure sequence of $\mathcal{M}(A, \mathbf{x}_0)$ is a periodic regime with period 1 and average commute time μ. Before, we said that the departure sequence of a max-plus model is entirely determined by its initial departure, but by theorem 1, we see that translating the initial departure and thus the entire sequence left both the commute and base sequences unchanged. This means that the behaviour of a departure sequence is not determined by its initial departure, but by the base of the initial departure. We will therefore always take our initial departure to be a base, i.e. $||\mathbf{x}_0||_{min} = 0$, to simplify calculations.[1]

We conclude this section with an important property of the max-plus model.

Theorem 2 *Any departure sequence of a max-plus model is causal and forgetful. In other words, the next entry in a sequence is dependent on the current entry, but not on any prior or future entries.*

This above result is important as it tells us that given any term in a departure sequence of a max-plus model, we can determine every term after the given term, without having to know any prior terms including the initial departure.

2.4. Finding eigenvalues: The Power Method

In this section, we will be solving the problem we formulated in section 2.1, by giving an algorithm for finding a finite eigenvalue. We will not go into specifics of how many eigenvalues a max-plus matrix has or discuss infinite eigenvalues, as this is not relevant to the formulated problem. We simply want to find the smallest finite eigenvalue of a given matrix and we will henceforth refer to the corresponding eigenvalue-eigenvector pair as the eigenvalue and the eigenvector of the given matrix.

Method 2 *Power Algorithm (Heidergott et al., 2006)*
Let A be the adjacency matrix of a strongly connected graph.

1. *Take an arbitrary initial departure $\mathbf{x}_0$ that has at least one finite element.*
2. *Calculate terms of the base sequence $(\lfloor\mathbf{x}(k)\rfloor)_{k\geq 0}$ until a periodic regime is reached. Take 2 repeating terms of the base sequence, the p'th and the q'th terms and let the magnitude of the translation from the p'th to the q'th term in the departure sequence be c.*
3. *The eigenvalue is $\lambda = c/(p-q)$, the average commute time.*
4. *The eigenvector is $\mathbf{v} = \bigoplus_{j=1}^{p-q}\left(\lambda^{\otimes(p-q-j)} \otimes \mathbf{x}(q+j-1)\right)$.*

Throughout this report, we will only consider the eigenvalue and eigenvector of a matrix found using the power algorithm. We verify that the algorithm works for the example given in figure 2.2, as applying the power algorithm to this examples yields the initial departure of sequence 2.1.

Henceforth, we will assume to always be working in max-plus algebra unless specified otherwise. This means max-plus matrices will just be called matrices, max-plus addition will just be called addition, etc. We will however distinguish between max-plus operations and 'plus-times' operations, by denoting the former as $\oplus, \otimes$ and the latter as $+, \times$ in order to maintain an intuitive element in calculations.

[1] This also means that when looking at very long departure sequences, we can study parts of the sequence by translating the entire system in such a way that the first entry is a base.

3

Switching Max-Plus

We can extend the max-plus model by allowing trains to speed up when desired. When allowing for this speed up, we call the model a switching max-plus model. At each time step, we can choose which trains to speed up and which to leave at their regular speed. In this chapter, we will discuss a simple example of a switching max-plus model, based on the example of chapter 2. We will then provide mathematical definitions and formulations for and related to switching max-plus models. We will conclude by showing how switching can help with reducing or resolving delays, as well as improving timetables. The contents of this chapter are based on the bachelor thesis 'Control of Delay Propagation in Railway Networks Using Max-Plus Algebra' (Hoekstra, 2020).

3.1. Example of a Switching System

We once again consider the example stated in section 2.1, as seen in figure 3.1.

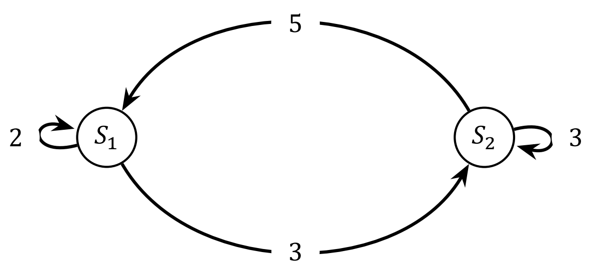


Figure 3.1: The communication graph of the example train network

We now allow the train on the inner arc with weight 5 to speed up, switching the weight to 4. We allow the train on the right most arc, labelled 3, to do the same so it can perform the commute in 2 time steps. Allowing these speed-ups, yields new communication graphs and adjacency matrices for this network:

$$A_0 = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix}$$

The first thing one might do, is see if one of these speed-ups allows for a faster timetable, i.e. a timetable with a lower average commute time. The answer in this case is yes, but doing so would remove the switching element from the system, as we would simply create a basic-max plus model using this optimal sped-up graph. The switching model allows for some flexibility in case things go wrong, such as delays. Using switching to optimise timetables would remove this added flexibility. Furthermore, increasing the speed of timetables in practice is not desirable. Many trains run on 1-hour or half-hour timetables, in part because most people's working day is split up into hours. Creating commutes where trains drive every 23 minutes for example, would not be practical for the average working person. Since we started with the network corresponding with adjacency matrix A_0, we call this adjacency matrix the standard adjacency matrix.

As previously said, the switching model allows for some flexibility in the system. In general, we can choose which trains to speed up and which to keep at their regular speed. This means at every time step there is a decision to be made: which adjacency matrix do we apply this time step? This yields the slightly altered recurrence relation

$$\begin{cases} \mathbf{x}(k+1) & = A(k) \otimes \mathbf{x}(k) \\ \mathbf{x}(0) & = \mathbf{x}_0 \end{cases}$$

Where $A(k) \in \{A_0, A_1, A_2, A_3\}$ is chosen at every time step. We can choose which adjacency matrix to apply in which time step ahead of time when planning timetables. When something goes wrong however, we have the freedom to choose another adjacency matrix better suited for that situation. Up to this point, we have been vague about what 'problems' may occur in the system. In section 3.3, we will give a concrete and notable example: Train delays.

3.2. Definitions

Before moving on to the example of train delays, we will formulate some definitions and form some intuition for how a switching max-plus model works on a mathematical level. We start off by formally introducing switching max-plus models.

Definition 8 *Let $\mathcal{A}$ be a set of $n \times n$ max-plus matrices and let $\mathbf{x}_0 \in \mathbb{R}^n_{max}$. We call $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0)$ the switching max-plus model of $\mathcal{A}$ with initial departure $\mathbf{x}_0$ when*

$$\mathcal{M}_S : \begin{cases} \mathbf{x}(k+1) & = A(k) \otimes \mathbf{x}(k) \\ \mathbf{x}(0) & = \mathbf{x}_0 \end{cases}$$

Where $A(k) \in \mathcal{A}$ is the adjacency matrix applied in time step k. We call $\mathcal{A}$ the adjacency class of $\mathcal{M}_S$

To determine which adjacency matrix will be applied in which time step, we can use a choice function. In practice, it can be useful to order the matrices in the adjacency class, both for the sake of intuition and implementation. In this ordering, we always call the first element, A_0, the standard adjacency matrix.

Definition 9 *Let $\mathcal{A} = \{A_0, \ldots, A_n\}$ be an adjacency class. A_0 is called the standard adjacency class of $\mathcal{A}$. Let J be a sequence containing the indices of the adjacency matrices $0, 1, \ldots n$ and let $J(k) = i$. Then A_i is called the expected adjacency matrix of time step k. If we apply $A_j, j \neq i$ in time step k, we say that we switched from matrix A_i to matrix A_j in time step k. We call J an index sequence.*

Through the ordering of the adjacency matrices, the choice function can simply map a situation (this can be a state, but may contain more information) to an index. We call the sequence of indices produced by the choice function the index sequence J. If this sequence is know ahead of time, we say that $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$. We illustrate this definition with an example.

Example 5 *Consider the switching max-plus system given in section 3.1, $\mathcal{M}_S(\mathcal{A}, \mathbf{x}_0), \mathcal{A} = (A_0, A_1, A_2, A_3), \mathbf{x}_0 = (1, 0)^T$. We apply matrices A_0 and A_3 in alternating fashion:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 8 \\ 8 \end{pmatrix} \rightarrow \begin{pmatrix} 13 \\ 11 \end{pmatrix} \rightarrow \begin{pmatrix} 15 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 21 \\ 19 \end{pmatrix} \rightarrow \ldots$$

The index sequence then is $J = (0, 3, 0, 3, 0, \ldots)$. The base sequence is:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 0 \end{pmatrix} \rightarrow \ldots$$

So we see that the sequence contains a periodic regime with period 2, onset 3 and average commute time 4. Upon further analysis, we can determine that the eigenvector of A_0 is $(1, 0)^T$ and the eigenvector of A_3 is $(0.5, 0)^T$, which are not the same. This means that in this departure sequence, we can not create a 1-periodic regime with the eigenvectors of both matrices. Since the index sequence is periodic however, we know that in our case:

$$\mathbf{x}(2k) = (A_3 \otimes A_0)^{\otimes k} \otimes \mathbf{x}(0)$$

This means that if we determine the eigenvectors of $A_3 \otimes A_0$, we can create a regular timetable using eigenvectors. However, since these eigenvectors correspond to the application of 2 matrices, we will not be creating a 1-periodic regime, but a 2-periodic regime. Indeed, when calculating the eigenvector of $A_3 \otimes A_0$, we get $(0, 1)^T$, which is exactly every second term in the base sequence after the onset of the periodic regime.

Theorem 3 *If the index sequence J of a switching max-plus model M_S on a strongly connected graph is m-periodic, then the eigenvector $\mathbf{v}$ of*

$$A = \bigotimes_{i=1}^{m} A_{J(m-i)}$$

induces an m-periodic regime with average commute time $\frac{\lambda}{m}$, where λ is the eigenvalue of A associated with $\mathbf{v}$.

We will generally denote the i'th entry of the index sequence J by $J(i)$ as J corresponds to a choice function projecting a time step $k = i$ to the index for that time step. The proof of the above theorem follows directly from the fact that $\mathcal{M}(A, \mathbf{v})$ has a 1-periodic regime with average commute time λ and the k'th term of its departure sequence is the $(m \times k)$'th term of the departure sequence of $\mathcal{M}_S(\mathcal{A}, \mathbf{v})$. The full proof of this theorem is given in appendix A. To give an intuitive interpretation of the result, we can not use the power algorithm to derive an eigenvector corresponding to a single time step of the system, but it can be used to derive an eigenvector for several consecutive time steps.

3.3. Delayed Trains

In chapter 2, we learned how to design an optimal, regular time table. At every time step, we knew exactly how long each train would take to reach their next destination. In practice however, trains do not always arrive at their allocated times, due to delays. Where in the basic max-plus model, we can only hope that the departure sequence converges back to its equilibrium sequence, the switching model allows us to actively intervene by speeding up trains.

Example 6 *Consider the train network with the adjacency class as described at the beginning of section 3.1 and in figure 3.1. When nothing goes wrong, we assume that the standard adjacency matrix is repeatedly applied. Choosing the initial departure $\mathbf{x}_0 = (1, 0)^T$, we arrive at the previously seen departure sequence*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 9 \\ 8 \end{pmatrix} \rightarrow \begin{pmatrix} 13 \\ 12 \end{pmatrix} \rightarrow \begin{pmatrix} 17 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 21 \\ 20 \end{pmatrix} \rightarrow \ldots \tag{3.1}$$

Induced by the initial departure and index sequence $J = (0, 0, 0, ...)$. Now let us assume that the train inbound for station S_1 at the first iteration experiences a delay of 2 time units:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5+2 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 10 \end{pmatrix} \to \begin{pmatrix} 15 \\ 13 \end{pmatrix} \to \begin{pmatrix} 18 \\ 18 \end{pmatrix} \to \begin{pmatrix} 23 \\ 21 \end{pmatrix} \to \dots \tag{3.2}$$

When subtracting sequence 3.1 from sequence 3.2, we see the delay propagation in each time step:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 2 \end{pmatrix} \to \begin{pmatrix} 2 \\ 1 \end{pmatrix} \to \begin{pmatrix} 1 \\ 2 \end{pmatrix} \to \begin{pmatrix} 2 \\ 1 \end{pmatrix} \to \dots$$

If we simply keep applying the standard adjacency matrix, this sequence will not return to its equilibrium. We therefore see if applying the other matrices in the adjacency class can resolve this issue. A naive but effective approach to doing this, is to simply apply every possible combination of adjacency matrices and seeing which combination restores the equilibrium as fast as possible. The fastest way to return to the equilibrium turns out to take 4 time steps. There are a total of 40 possible ways to restore equilibrium in 4 time steps, one of which is the following:

$$A_1^{\otimes 3} \otimes A_0 \otimes \begin{pmatrix} 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 21 \\ 20 \end{pmatrix}.$$

This results in the delayed departure sequence

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5+2 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 10 \end{pmatrix} \to \begin{pmatrix} 14 \\ 13 \end{pmatrix} \to \begin{pmatrix} 17 \\ 17 \end{pmatrix} \to \begin{pmatrix} 21 \\ 20 \end{pmatrix} \to \dots$$

With corresponding index sequence $J = (0, 0, 1, 1, 1, 0, 0, ...)$. We clearly see that the fifth term of this delayed sequence coincides with the fifth term of the timetable sequence. The resulting delay propagation sequence now becomes

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 2 \end{pmatrix} \to \begin{pmatrix} 1 \\ 1 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \dots$$

In light of the above example, we provide some more definitions

Definition 10 *If a trains do not leave their outbound station at the expected time $\mathbf{x}(k)$, we say that that station is delayed. We call the actual departure time vector $\tilde{\mathbf{x}}(k)$ the delayed departure and the difference $\tilde{\mathbf{x}}(k) - \mathbf{x}(k) = \mathbf{d}(k)$ the delay state of the system at time step k. We call the sequence $(\mathbf{d}(k))_{k\geq 0}$ the delay propagation sequence of the system, or delay sequence for short.*

Trivially, the delay sequence of the departure sequence with itself contains only 0-vectors.

Definition 11 *Let $(\mathbf{d}(k))_{k\geq 0}$ be a delay sequence of $\mathcal{M}_S$. We call the first k such that $\mathbf{d}(k) \neq \mathbf{0}$ the onset of the delay and we call this $\mathbf{d}(k)$ the initial state delay.*

Definition 12 *Let $(\mathbf{x}(k))_{k\geq 0}$ and $(\tilde{\mathbf{x}}(k))_{k\geq 0}$ be the departure sequence and a delayed departure sequence of $\mathcal{M}_S$. We say that $(\tilde{\mathbf{x}}(k))_{k\geq 0}$ converges to $(\mathbf{x}(k))_{k\geq 0}$ if:*

$$\exists N \in \mathbb{N} : \forall k \geq N : \tilde{\mathbf{x}}(k) = \mathbf{x}(k).$$

We call the smallest such N the resolution time of the delayed departure sequence.

Note that by definition 10, sequence 3.2 is the sequence of delayed departures, which we will call the delayed departure sequence.

It is important to note that in the current statement of the model, delays do not occur. We therefore need to change the model in order to include delays. There are 2 ways to do this, we can simply add the delay state to the recurrence relation or we can change the adjacency matrix applied in a time step to reflect the delay:

$$\mathbf{x}(k+1) = A_i \otimes \mathbf{x}(k) + \delta(k) \qquad \text{or} \qquad \mathbf{x}(k+1) = \tilde{A}_i \otimes \mathbf{x}(k).$$

The first way keeps things more intuitive and easy to implement into code, but it does require the use of a plus-times addition of 2 vectors[1]. We call the vector $\delta(k)$ the delay term of the model. Notice that if there is a delay, then $\delta(k) = \mathbf{d}(k+1)$. The latter way introduces a less intuitive element, $\tilde{A}_i$, which we will henceforth call a delayed adjacency matrix of A_i, but it does match the max-plus structure used thus far. In order to not needlessly complicate the intuition of the model, we will use the former notation, despite not maintaining the max-plus form.

We should also note that delays can stack, that is to say that new delays can affect the delay propagation of a prior delay. When this happens, a completely new delayed departure sequence can be computed using $\mathbf{x}(k+1)$ as a starting vector where k is the onset of the last delay. Since max-plus systems are forgetful, the manner in which the resulting delay was induced does not affect the induced delayed departure sequence.

Definition 13 *Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0)$ and let $(\tilde{\mathbf{x}}(k))_{k\geq 0}$ be a delayed departure sequence of $\mathcal{M}_S$. If $\tilde{\mathbf{x}}(k) = \mathbf{x}(k)$ and $\delta(k) \neq \mathbf{0}$, then $\delta(k)$ is called a simple delay.*

To conclude this section, we give an example using the defined concepts.

Example 7 *Consider the following departure sequence:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 1 \end{pmatrix} \to \begin{pmatrix} 3 \\ 3 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 6 \\ 6 \end{pmatrix} \to \begin{pmatrix} 8 \\ 7 \end{pmatrix} \to \dots$$

And consider the following delayed departure sequence:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 1 \end{pmatrix} \to \begin{pmatrix} 4 \\ 3 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 7 \\ 6 \end{pmatrix} \to \begin{pmatrix} 8 \\ 7 \end{pmatrix} \to \dots$$

This sequence has 2 delays, $\delta(1) = (1,0)^T, \delta(3) = (1,0)^T$ with onsets 2 and 4 respectively. The delayed departure sequence induced by $\delta(1)$ is resolved at time step $k = 2$ and the delayed departure sequence induced by $\delta(3)$ is resolved at time step $k = 4$.

Generally, looking at several delays in the same sequence can complicate matters. Because of this, we will treat every delay separately unless stated otherwise. If delays influence each other, we will combine them by taking the time step immediately after the onset of the last delay $\delta(k) \neq \mathbf{0}$, as previously stated above definition 13. It is important to note that we can generally not predict when delays will happen. This means we will always make decisions in the system without taking possible future delays into account. Concretely, in example 7, the delayed departure sequence induced by $\delta(1)$ does not include the delay induced by $\delta(3)$ since at time step 1 it was not yet known that the delay at times step 3 would occur.

[1] An element wise max-plus multiplication of vectors would have the same effect, but this operation has not been used in any other instances, so introducing it would not be practical.

3.4. Timetable Improvements

In addition to resolving delays, switching max-plus systems can also be used to improve departure sequences, which are referred to as timetables in practice. By speeding up and slowing down trains, more convenient time tables can be created for regular commuters.

Example 8 *Consider the example communication graph in figure 3.1 with the following adjacency matrices:*

$$A_0 = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 5 \\ 2 & 3 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 6 \\ 3 & 3 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 2 & 6 \\ 2 & 3 \end{pmatrix}$$

With initial departure $\mathbf{x}_0 = (1,0)^T$. If we simply repeatedly apply A_0, we get the 1-periodic regime:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 9 \\ 8 \end{pmatrix} \rightarrow \begin{pmatrix} 13 \\ 12 \end{pmatrix} \rightarrow \begin{pmatrix} 17 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 21 \\ 20 \end{pmatrix} \rightarrow \dots$$

But let now assume that at time steps 9 a plane arrives at the airport connected to station S_1 and at time steps 11 a plane departs near station S_2 and there are a lot of passengers that want to transfer between these two planes. This means there are a lot of passengers that will take the train at 9 in station S_1 and they will arrive in S_2 at 12, just one time unit too late for their plane. We can choose to speed up the corresponding train during these time steps by applying adjacency matrix A_1:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 9 \\ 8 \end{pmatrix} \rightarrow \begin{pmatrix} 13 \\ 11 \end{pmatrix} \rightarrow \begin{pmatrix} 16 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 21 \\ 19 \end{pmatrix} \rightarrow \dots$$

So we see these passengers leaving S_1 at 9 make it to S_2 by 11, but the regime has completely changed. We can now choose to slow down the same train now driving back to station S_1 by applying matrix A_2:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 9 \\ 8 \end{pmatrix} \rightarrow \begin{pmatrix} 13 \\ 11 \end{pmatrix} \rightarrow \begin{pmatrix} 17 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 21 \\ 20 \end{pmatrix} \rightarrow \dots$$

So we see that switching can be used to construct more intricate departure sequences that are tailored to a specific situation. In theory however, the above process also describes a sort of delay, namely a predicted, negative delay. As such, we can simply regard such situations as delay situations.

4

Delay Problems

Now that we have laid the foundation of switching max-plus models, we can formally introduce the problem we seek to research in this report: How can we optimally resolve delays in train networks.

Problem 1 *The Delay Problem:*
Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ be a switching max-plus model. Given a delayed departure $\tilde{\mathbf{x}}(m)$, give an index sequence $\tilde{J}$ such that the delayed departure sequence $(\tilde{\mathbf{x}}(k))_{k \geq m}$ induced by $\tilde{\mathbf{x}}(m)$ and $\tilde{J}$ converges to the departure sequence of $\mathcal{M}_S$ as quickly as possible, i.e. with the smallest possible resolution time.

Since we established that we will never act on delays that have not yet happened, we will always take the elements of $\tilde{J}$ before the onset of the delay to be the same as J. We will later come back to the formulation of this problem and make some changes to refine our search for the index sequence $\tilde{J}$.

In this chapter, we will start analysing this problem with the concepts we have defined thus far. We will also introduce some new definitions to aid our analyses. We will start by resolving some example delays using an exhaustive method. We will then analyse the resulting optimum to see if we can spot patterns. After this, we will attempt to come up with criteria for finding index sequences using more elaborate methods. We will conclude this chapter by introducing a more drastic measure that can be used when delays become unmanageable.

4.1. Solving the Problem: Every Possibility

In this section, we will look into one method for solving the delay problem. This method involves exhaustively trying every possible combination of index sequences $\tilde{J}$ and seeing which sequence works. We will apply this method to an example before formulating the algorithm used. We will then put an extra condition on the solutions of problem 1 to narrow down our search. We will conclude this section by showing a restriction of this method to illustrate why it can not always be used in practice.

4.1.1. The Combinatorial Method

We will once again consider the departure and delayed departure sequences from example 6.

Example 9 *Consider the switching model $\mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ where*

$$A_0 = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix}$$

$\mathbf{x}_0 = (1,0)^T$ *and* $J = (0,0,0,...)$ *This induces the following departure sequence:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 20 \end{pmatrix} \to \dots$$

Now assume that some delay was caused with initial delay

$$\delta(0) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}.$$

Problem 1 dictates that we try to find $\tilde{J}$ *of the smallest size* m *such that:*

$$\bigotimes_{i=1}^{m} (A_{\tilde{J}(m-i)}) \otimes \mathbf{x}(1) = \mathbf{x}(m+1). \tag{4.1}$$

We try every combination and find that the smallest such m *is 4. A solution index sequence is:*

$$\tilde{J} = 0,0,1,1,1,0,0,...$$

which induces the following delayed departure sequence:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 7 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 10 \end{pmatrix} \to \begin{pmatrix} 14 \\ 13 \end{pmatrix} \to \begin{pmatrix} 17 \\ 17 \end{pmatrix} \to \begin{pmatrix} 21 \\ 20 \end{pmatrix} \to \dots$$

Which indeed converges to the departure sequence. We notice that since the onset of the delay is 1 and its resolution time is 5, that the only part of $\tilde{J}$ *that resolved the delay, is the four entries after the first entry:* $0,1,1,1$.

In light of the above example, we give the following definition.

Definition 14 *Let* $(\tilde{\mathbf{x}}(k))_{k\geq 0}$ *be a delayed sequence with only one initial delay* $\delta(p)$*. Let* $\tilde{J}$ *be an index sequence that resolves the delay with resolution time* $k = q$*. We call the finite sequence* $\tilde{R} = \big(\tilde{J}(p+1), \tilde{J}(p+2), \dots, \tilde{J}(q-1)\big)$ *the resolution sequence of the delay and the index sequence.*

In this example, we have also stated problem 1 in its alternative form in equation 4.1.

Problem 2 *The Delay Problem*
Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ *be a switching max-plus model. Given a delayed departure* $\tilde{\mathbf{x}}(m)$*, give an index sequence* $\tilde{J}$ *of minimal length* m *such that*

$$\bigotimes_{i=1}^{m} (A_{\tilde{J}(m-i)}) \otimes \mathbf{x}(1) = \mathbf{x}(m+1).$$

In example 9, we tried every possible index sequence to find a solution to problem 1. We can formulate this procedure in the following method.

Method 3 *The Combinatorial Method*
Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ be a switching max plus model. Let $\tilde{\mathbf{x}}(m)$ be a delayed departure.

1. *Take $n = 0$.*
2. *For any $R \in \mathcal{A}^n$:*
   (a) *Calculate $\tilde{\mathbf{x}}(m+n) = \bigotimes_{i=1}^{n} \left(A_{J(n-i)}\right) \otimes \tilde{\mathbf{x}}(m)$*
   (b) *If $\tilde{\mathbf{x}}(m+n) = \mathbf{x}(m+n)$, then R is a resolution sequence $\tilde{R}$ of the delay with resolution time $m+n$.*
3. *If no $\tilde{R}$ is found, increment n by 1 and return to step 2.*

Repeat until a stopping criteria is met.

It is clear to see that since this method calculates every possible outcome, we will always arrive at the optimal solution to problem 1, if one exists. In practice, whenever there is a delay, we will take the entries in both the departure and the delayed departure sequences corresponding to the onset of the delay, translate them such that one of them is a base and create the sequences induced by these elements. This also ensures that the resolution sequence of the index sequence $\tilde{J}$ always appears at the start of $\tilde{J}$. Because of this, we will often call n the resolution time of the delay and $m+n$ the resolution time of the delay sequence, as dictated by definition 12.

Example 10 *Consider the same switching model as in example 9 with departure and delayed departure sequences respectively*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 3 \end{pmatrix} \to \begin{pmatrix} 5 \\ 5 \end{pmatrix} \to \begin{pmatrix} 7 \\ 8 \end{pmatrix} \to \begin{pmatrix} 10 \\ 10 \end{pmatrix} \to \begin{pmatrix} 12 \\ 13 \end{pmatrix} \to \dots$$
$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 3 \end{pmatrix} \to \begin{pmatrix} 5 \\ 5 \end{pmatrix} \to \begin{pmatrix} 7 \\ 8 \end{pmatrix} \to \begin{pmatrix} 11 \\ 11 \end{pmatrix} \to \begin{pmatrix} 13 \\ 13 \end{pmatrix} \to \dots$$

and corresponding index sequences respectfully

$$J = 0, 0, 0, 0, 0, 0, \dots$$
$$\tilde{J} = 0, 0, 0, 0, 1, 1, \dots$$

Since the delay onset happens later in the sequence, the system is forgetful and sequence behaviour does not change due to translations, we can simply study the following pair of sequences and get the same result:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 3 \end{pmatrix} \to \begin{pmatrix} 5 \\ 5 \end{pmatrix} \to \begin{pmatrix} 7 \\ 8 \end{pmatrix} \to \begin{pmatrix} 10 \\ 10 \end{pmatrix} \to \begin{pmatrix} 12 \\ 13 \end{pmatrix} \to \dots$$
$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 1 \end{pmatrix} \to \begin{pmatrix} 3 \\ 3 \end{pmatrix} \to \begin{pmatrix} 5 \\ 5 \end{pmatrix} \to \begin{pmatrix} 7 \\ 8 \end{pmatrix} \to \begin{pmatrix} 11 \\ 11 \end{pmatrix} \to \begin{pmatrix} 13 \\ 13 \end{pmatrix} \to \dots$$

With index sequences respectfully

$$J = 0, 0, 0, 0, 0, 0, \dots$$
$$\tilde{J} = 1, 1, 0, 0, 0, 0, \dots$$

So we see that the index sequence of the delayed departures is simply shifted to the right without changing its structure.

4.1.2. The Minimal Solution

As mentioned previously in example 6, the solution found in example 9 is not unique. There are a total of 40 possible resolution sequences of length 4. This means we can choose among the solution which one we deem the 'most optimal'. Since we want delays to be resolved as quickly as possible however, we will always consider the minimum resolution time our primary criterion. In this subsection, we will discuss which possible secondary criteria we can devise for our solution.

The model of example 9 contains the following adjacency matrices:

$$A_0 = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix}$$

Where A_0 is the standard matrix. We will therefore compare the other matrices to A_0 for our secondary criterion. A realistic property of the secondary criterion is that we want to have as few speed ups as possible. Unnecessary speed ups may cause excess noise or wear down the rails. This gives the following secondary criterion:

'Find $\tilde{J}$ such that $\sum \#\{(i,j) : [A_{\tilde{J}(k)}]_{i,j} \neq [A_{J(k)}]_{i,j}\}$ is as small as possible.'

Another reasonable criterion is that we do not care so much about the amount of speed ups, but rather the total magnitude of the speed ups. This means we find minor noise pollution in several regions less problematic than major noise pollution in one region. The same argument can be used for wearing down the tracks.

'Find $\tilde{J}$ such that $\sum ||A_{\tilde{J}(k)} - A_{J(k)}||_1$ is as small as possible.'

Where $||A_{\tilde{J}(k)} - A_{J(k)}||_1$ is the sum of the absolute element differences of the matrices. If we care more about maintaining logistic integrity of the network, instead of impact on the environment, a reasonable criterion could be that we want to cause as little deviation from the regular index sequence as possible.

'Find $\tilde{J}$ such that $\#\{i : \tilde{J}(i) \neq J(i)\}$ is as small as possible.'

In this section, we will use the second of these secondary criteria as it takes into account both the occurrence of a speed up and the magnitude of this speed up.

Problem 3 *The Minimal Delay Problem*
Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ be a switching max-plus model. Given a delayed departure $\tilde{\mathbf{x}}(m)$, give a smallest resolution sequence $\tilde{R}$ such that

$$\sum ||A_{\tilde{R}(k)} - A_{J(k)}||_1$$

Is as small as possible.

When applying this secondary criterion to example 9, we see that from the original 40 possible solutions, there are 2 solutions for which $\sum ||A_{\tilde{R}(k)} - A_{J(k)}||_1 = 3$, which is the minimum. This means there are 2 optimal solutions:

$$\tilde{R}_1 = 0, 1, 1, 1$$
$$\tilde{R}_2 = 0, 3, 0, 1$$

We call these solutions to problem 3, minimal solutions to problem 1.

The minimal delay problem adds an extra layer onto the delay problem and as such, should only be considered once the latter problem has been properly solved. Because of this, we will not devote too much attention to the minimal problem, as to not get ahead of ourselves. The reason for providing the formulation of the problem, is that in some networks, finding a minimal delay resolution is more important than in others and so we will be discussing some minimal delay resolution method, but we will not be analysing them in great detail.

4.1.3. Computational Restrictions

We have seen that the combinatorial method can always find the optimal solutions to both the delay and the minimal delay problems if they exist. Based on this, one could say the this method is perfect and does not need to be altered in any way. Unfortunately however, though we can be sure that the combinatorial method will always find the optimal solution, the speed at which it does so turns out to be very volatile. For small examples, the computation time does not exceed a second, but for larger systems, this duration can become unmanageable. To better be able to discuss this issue, we look into the time complexity of the algorithm.

The algorithm iterates over values for the resolution of the delay until a resolution sequence is found. Let $N(s)$ be the time complexity of the iteration with value $n = s$, then the time complexity of the combinatorial algorithm is

$$T(CM) = \mathcal{O}(N(0) + N(1) + N(2) + \ldots + N(n))$$

At every iteration, every element of the cartesian product $\mathcal{A}^n$ is calculated and applied, meaning the time complexity of an iteration $N(s)$ is

$$N(s) = \mathcal{O}((\#\mathcal{A})^s)$$

This yields the time complexity

$$T(CM) = \mathcal{O}((\#\mathcal{A})^m)$$

where m is the length of a minimal resolution sequence. If this m is known, then the algorithm has a polynomial time complexity of order m in the size of the adjacency class $\mathcal{A}$. In practice however, it is unlikely that we know m ahead of time, so the algorithm has an exponential time complexity in m. Furthermore, we can not ensure that the delayed departure sequence converges to the departure sequence, so it is possible that the algorithm will not produce a result at all, but there is no way to find out if this is the case using the combinatorial algorithm.

Example 11 *Consider a switching max-plus model such that there are 5 connections we can speed up by 1 time unit. Since we choose for every connection if we speed them up or not, there are* 32 *possible adjacency matrices, so the size of the adjacency class is* $\#\mathcal{A} = 32$*. Suppose a delay is created that can be resolved in 10 time steps, but not less. Then the time complexity of the combinatorial algorithm is*

$$\begin{aligned} T(CM(\mathcal{A}, \tilde{\mathbf{x}}(m))) &\approx C \times 32^{10} \\ &\approx C \times 1.126 \times 10^{15} \end{aligned}$$

for some C*.*

If we assume C to be very small, like $C = 10^{-10}s$, this would still leave us with a computation time of approximately 31 hours. In practice, C may be much smaller, but at the same time, the adjacency matrices may by much larger. Thus national train networks with thousands of connection could not resolve delays in this way.[1]

4.2. Sub-Optimal Methods

In order to combat the unreasonable computation times of the combinatorial method, we will attempt to devise iterative methods. This means we want to create methods that do not attempt to provide a full

[1] The same calculation for 30 connections and $C = 10^{-30}s$ yields a computation time of more than 6×10^{52} years.

solution right-away, but instead calculate partial results which they use to determine the best course of action for the next time step. We will derive and formally introduce two such methods: The greedy delay resolution method and the composite greedy delay resolution method. After having formulated them, we will look at their computation time.

4.2.1. The Greedy Delay Resolution Method

In order to derive an iterative method, we will dissect each time step to see what the best course of action is for this step.

Example 12 *Consider the departure sequence and initial delay of example 9:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 9 \\ 8 \end{pmatrix} \rightarrow \begin{pmatrix} 13 \\ 12 \end{pmatrix} \rightarrow \begin{pmatrix} 17 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 21 \\ 20 \end{pmatrix} \rightarrow \dots$$

$$\delta(0) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}.$$

This means the first delayed departure is $\tilde{\mathbf{x}}(1) = (7,4)^T$ with corresponding expected departure $\mathbf{x}(1) = (5,4)^T$. We translate both departures, so one of them is a base, which yields the same departure sequence as above and the initial delayed departure $\tilde{\mathbf{x}}(0) = (3,0)^T$. We now look at each possible delay propagation by applying each adjacency matrix in the class separately

$$A_0 \otimes \tilde{\mathbf{x}}(0) = \begin{pmatrix} 5 \\ 6 \end{pmatrix} \qquad A_1 \otimes \tilde{\mathbf{x}}(0) = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

$$A_2 \otimes \tilde{\mathbf{x}}(0) = \begin{pmatrix} 5 \\ 6 \end{pmatrix} \qquad A_3 \otimes \tilde{\mathbf{x}}(0) = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

We see at all matrices yield the same result, so we choose the index i such that $||A_i - A_0||_1$ is minimised, so $\tilde{R}(0) = 0$. We now apply the same procedure to $\tilde{\mathbf{x}}(1) = (5,6)^T$:

$$A_0 \otimes \tilde{\mathbf{x}}(1) = \begin{pmatrix} 11 \\ 9 \end{pmatrix} \qquad A_1 \otimes \tilde{\mathbf{x}}(1) = \begin{pmatrix} 10 \\ 9 \end{pmatrix}$$

$$A_2 \otimes \tilde{\mathbf{x}}(1) = \begin{pmatrix} 11 \\ 8 \end{pmatrix} \qquad A_3 \otimes \tilde{\mathbf{x}}(1) = \begin{pmatrix} 10 \\ 8 \end{pmatrix}$$

When comparing these delayed departures to the expected departure $\mathbf{x}(2)$, we notice that some delayed departures show promising results.

We can intuitively see that some delayed departures are 'closer' to the expected departures than others. Because we have not defined what 'close' means however, we have no way to quantify which delayed departure is actually closer. As before, we can define the distance between two states in several ways. We will limit ourselves to the following well known metrics:

- $||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum (x_i - y_i)^2}$, the euclidean metric.
- $||\mathbf{x} - \mathbf{y}||_1 = \sum |x_i - y_i|$, the sum of absolute element differences.
- $||\mathbf{x} - \mathbf{y}||_{max} = \max\{|x_i - y_i|\}$, the maximum absolute element difference.

Each of these metrics have benefits and downsides and must therefore be chosen on a case-by-case basis for practical applications. In this report, we will be using the last of these three as our primary metric and the second as our secondary metric. Armed with these new notions of distance, we can continue example 12.

Continuation of example 12 *We see that matrices A_1 and A_3 both minimise the distance $||\tilde{\mathbf{x}}(2) - \mathbf{x}(2)||_{max}$, but A_1 is further away according to $|| \cdot ||_1$, so we choose A_3, thus $\tilde{R}(1) = 3$. In the next* *iteration, we get*

$$A_0 \otimes \tilde{\mathbf{x}}(2) = \begin{pmatrix} 13 \\ 13 \end{pmatrix} \qquad A_1 \otimes \tilde{\mathbf{x}}(2) = \begin{pmatrix} 12 \\ 13 \end{pmatrix}$$

$$A_2 \otimes \tilde{\mathbf{x}}(2) = \begin{pmatrix} 13 \\ 13 \end{pmatrix} \qquad A_3 \otimes \tilde{\mathbf{x}}(2) = \begin{pmatrix} 12 \\ 13 \end{pmatrix}$$

We now see that all of these delayed departures have the same max-distance to $\mathbf{x}(3)$. The departures corresponding to A_0 and A_2 are closer according to the 1-norm however, so we choose on of them. Since we also want to minimise the value $||A_i - A_{J(2)}||_1$, we choose the matrix with the smallest distance to A_0, so $\tilde{R}(2) = 0$. Continuing this procedure gives us the resolution sequence $\tilde{R} = (0, 3, 0, 1)$, which is indeed a minimal resolution sequence and is in fact a solution to the minimal delay problem.

We see that when using just the max-norm, a situation arose where two decisions were evaluated as equal. In order to combat this, we used the 1-norm to break the tie. It is possible for more such ties to occur, but in order to avoid complexity, we will not formulate other tie breakers, instead choosing one of the equal choices based on some criteria[2].

Method 4 *Greedy Delay Resolution Method*
Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ be a switching max-plus model. Let $\tilde{\mathbf{x}}(m)$ be a delayed departure.

1. *Let $n = 0$*
2. *If $\tilde{\mathbf{x}}(m+n) = \mathbf{x}(m+n)$, we are done.*
3. *For any $A_i \in \mathcal{A}$, calculate $\tilde{\mathbf{x}}_i(m+n+1) = A_i \otimes \tilde{\mathbf{x}}(m+n)$.*
   (a) *From the resulting delayed departures, choose the delayed departure(s) $\tilde{\mathbf{x}}_i(m+n+1)$ such that $||\tilde{\mathbf{x}}_i(m+n+1) - \mathbf{x}(m+n+1)||_{max}$ is minimised.*
   (b) *From the resulting delayed departures, choose the delayed departure(s) $\tilde{\mathbf{x}}_i(m+n+1)$ such that $||\tilde{\mathbf{x}}_i(m+n+1) - \mathbf{x}(m+n+1)||_1$ is minimised.*
4. *Choose one of the resulting delayed departures $\tilde{\mathbf{x}}_j(m+n+1)$ based on some criteria and set $\tilde{\mathbf{x}}(m+n+1) = \tilde{\mathbf{x}}_j(m+n+1)$ and $\tilde{R}(n) = j$.*
5. *Increment n by 1 and return to step 2*

The resulting n and $\tilde{R}$ are the resolution time and resolution sequence respectively.

This method has 2 major benefits compared to the combinatorial method. The first is that the computation time is vastly reduced, which we will confirm later on. The second is that while the combinatorial method can only produce the resolution time and sequence, the greedy delay resolution method can also produce the intermediary results obtained at each time step. This means we can study the behaviour of the delay sequence as we apply the method, which makes it easier to see if the delay will ever be resolved at all.

Theorem 4 *Let $\mathcal{A}$ be an adjacency class, $(\mathbf{x}(k))_{k\geq 0}$ and $(\mathbf{y}(k))_{k\geq 0}$ departure sequences induced by J, a choice function generating an index sequence and some initial departures $\mathbf{x}(0)$ and $\mathbf{y}(0)$ respectively. Suppose that $J = J(\mathbf{x})$, so the choice function only depends on the current state, then the following hold:*

[2] If we want to find a minimal solution, choose an index such that $||A_j - A_{J(n)}||_1$ is minimised.

- *If there is a time step n where the departure times $\mathbf{x}(n) = \mathbf{y}(n)$, then the two departure sequences are equal in every time step after n.*

$$\text{If } \exists n \in \mathbb{N} : \mathbf{x}(n) = \mathbf{y}(n) \text{ then } \forall k \geq n : \mathbf{x}(k) = \mathbf{y}(k)$$

- *Suppose $(\mathbf{x}(k))_{k\geq 0}$ and $(\mathbf{y}(k))_{k\geq 0}$ enter s-periodic regimes with onset n and m respectively. Let $k \geq \max(n, m)$, if $\mathbf{x}(k) \neq \mathbf{y}(k)$, then the two departure sequences at no point coincide.*

We provide a proof of this theorem in appendix A. Upon close inspection, we see that the choice function resulting from the greedy delay resolution method fulfils the conditions of this theorem.

This above result shows us that if a switching max-plus delayed departure sequence ever enters into a periodic regime that does not coincide with the departure sequence, then the delay can not be resolved using that choice function. We will illustrate this theorem with an example, whilst simultaneously showing a severe downside of the greedy delay resolution method.

Example 13 *Consider the switching max-plus model $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ with adjacency matrices, initial departure and index sequence respectively*

$$A_0 = \begin{pmatrix} \varepsilon & 5 \\ 5 & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} \varepsilon & 3 \\ 3 & \varepsilon \end{pmatrix}$$

$$A_2 = \begin{pmatrix} \varepsilon & 2 \\ 2 & \varepsilon \end{pmatrix}$$

$$\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad J = 0, 0, 0, 0, \ldots$$

which corresponds to the following communication graph with variable commute times:

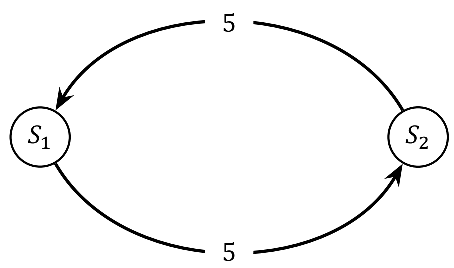


Figure 4.1: The communication graph of the example train network

The departure sequence induced by $\mathbf{x}_0$ and J is:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 5 \end{pmatrix} \to \begin{pmatrix} 10 \\ 10 \end{pmatrix} \to \begin{pmatrix} 15 \\ 15 \end{pmatrix} \to \begin{pmatrix} 20 \\ 20 \end{pmatrix} \to \begin{pmatrix} 25 \\ 25 \end{pmatrix} \to \ldots$$

Now consider the delay $\delta = (4, 4)^T$. We use the greedy delay resolution method to resolve this delay. This produces the following delayed departure sequence:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 4 \\ 4 \end{pmatrix} \to \begin{pmatrix} 6 \\ 6 \end{pmatrix} \to \begin{pmatrix} 11 \\ 11 \end{pmatrix} \to \begin{pmatrix} 16 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 21 \end{pmatrix} \to \begin{pmatrix} 26 \\ 26 \end{pmatrix} \to \ldots$$

along with the resolution sequence $\tilde{R} = (2, 0, 0, 0, \ldots)$. It is clear to see that by theorem 4, this delay will never be resolved, the departure sequence and the delayed departure sequence both enter into 1-periodic regimes with different elements. Upon close inspection however, it is easy to see that:

$$\tilde{\mathbf{x}}(3) = A_1 \otimes A_1 \otimes \tilde{\mathbf{x}}(0) = \begin{pmatrix} 10 \\ 10 \end{pmatrix} = \mathbf{x}(3)$$

Which implies that by theorem 4, $(\tilde{\mathbf{x}}(k))_{k\geq 3} = (\mathbf{x}(k))_{k\geq 3}$*, So the delay can be resolved, but this solution is not found by the greedy method.*

The above example illustrates that even when a simple resolution exists, the greedy algorithm can not always find it. It is however not too difficult to show that a solution will not be reached, meaning the combinatorial method can then be used as an alternative for simple delay problems. Another downside of the greedy delay resolution method is that even if we set the criteria in step 4 to minimise $||A_j - A_{J(n)}||_1$, we can not guarantee that if the method produces a solution to the delay problem, this is a minimal solution. The combinatorial method can ensure this.

4.2.2. The Composite Greedy Delay Resolution Method

We have seen that a major detriment of the combinatorial method is its exponential time complexity in the resolution time and a major detriment of the greedy delay resolution method is its inability to consistently produce a result. We will now introduce a resolution method that combines both methods to counteract their detriments.

The intuition behind the composite greedy delay resolution method is that instead of taking the best course of action per time step, we combine several time steps and determine the best course of action over all of them. Determining the best course of action over several time steps can be done using the combinatorial method. Using the same distance criteria as earlier, we determine which combination of adjacency matrices gets the delayed departure as close to the expected departure as possible.

Method 5 p*-Composite Greedy Delay Resolution Method*
Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ *be a switching max-plus model. Let* $\tilde{\mathbf{x}}(m)$ *be a delayed departure.* p *is a natural number.*

1. *Let* $n = 0$
2. *Use the combinatorial method to determine a finite sequence* $\tilde{R}_n$ *of length* p*. such that:*

$$\tilde{\mathbf{x}}(m + (n+1) \times p) = \bigotimes_{i=1}^{p} \left(A_{\tilde{R}_n(p-i)}\right) \otimes \tilde{\mathbf{x}}(m + n \times p)$$

   (a) $||\tilde{\mathbf{x}}(m + (n+1) \times p) - \mathbf{x}(m + (n+1) \times p)||_{max}$ *is minimised.*
   (b) $||\tilde{\mathbf{x}}(m + (n+1) \times p) - \mathbf{x}(m + (n+1) \times p)||_1$ *is minimised.*

3. *If* $\exists q \leq p : \tilde{\mathbf{x}}(m + n \times p + q) = \mathbf{x}(m + n \times p + q)$*, shorten* $\tilde{R}_n$ *to its first* q *entries. We are done.*
4. *Choose one of the resulting finite sequences* $\tilde{R}_n$ *based on some criteria.*
5. *Increment* n *by 1 and return to step 2*

$n \times p + q$ *is the resolution time and the concatenation of all* $\tilde{R}_i$ *is the resolution sequence.*

For small resolution times, particularly resolution times smaller than p, this method is the same as the combinatorial method. For larger resolution times however, this method can harshly reduce the computation time needed to resolve a delay.

Example 14 *Consider the same switching model as in example 13, but this time with the delay* $\delta = (100, 100)^T$*. It is easy to show that every time step, we can reduce the delay by at most 3. This means the delay will take a minimum of 33 time steps to be resolved. The estimated time for using the combinatorial method to solve this problem is approximately a millennium. When using the greedy delay resolution method, we encounter the same problem as before that the delay is never resolved. Now using the 5-composite greedy delay resolution method, we see that as expected, the delay is* *resolved with resolution time 34:*

$$\begin{pmatrix} S_1 \\ S_2 \\ k \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \to \dots \to \begin{pmatrix} 150 \\ 150 \\ 30 \end{pmatrix} \to \begin{pmatrix} 155 \\ 155 \\ 31 \end{pmatrix} \to \begin{pmatrix} 160 \\ 160 \\ 32 \end{pmatrix} \to \begin{pmatrix} 165 \\ 165 \\ 33 \end{pmatrix} \to \begin{pmatrix} 170 \\ 170 \\ 34 \end{pmatrix}$$

$$\begin{pmatrix} S_1 \\ S_2 \\ k \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \to \dots \to \begin{pmatrix} 160 \\ 160 \\ 30 \end{pmatrix} \to \begin{pmatrix} 163 \\ 163 \\ 31 \end{pmatrix} \to \begin{pmatrix} 166 \\ 166 \\ 32 \end{pmatrix} \to \begin{pmatrix} 168 \\ 168 \\ 33 \end{pmatrix} \to \begin{pmatrix} 170 \\ 170 \\ 34 \end{pmatrix}$$

Where the resolution sequence is:

$$\begin{pmatrix} J \\ k \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \dots, \begin{pmatrix} 2 \\ 29 \end{pmatrix}, \begin{pmatrix} 2 \\ 30 \end{pmatrix}, \begin{pmatrix} 1 \\ 31 \end{pmatrix}, \begin{pmatrix} 1 \\ 32 \end{pmatrix}, \begin{pmatrix} 2 \\ 33 \end{pmatrix}, \begin{pmatrix} 2 \\ 34 \end{pmatrix}$$

The computation time for this method was less than a second.

By increasing p, more accurate results can be derived, but the computation time increases. In the extreme where $p = 1$ we have the regular greedy delay resolution method and in the extreme where p is equal to the resolution time, the method is the same as the combinatorial method. Note that it is still possible for this method to fail for values of p that are too small, but the larger we set p, the less likely this is to happen. We also see that this method, like the regular greedy method, is not guaranteed to produce a minimal solution even if a solution is reached, which is why both methods are functional, but sub-optimal methods.

4.2.3. Time Complexity

Now that we have formulated these sub-optimal method to reduce the computation time, we determine their time complexities to determine how big this improvement really is.

We start with the greedy delay resolution method. Let m be the resolution time of a delay using the greedy method. Let $N(s)$ be the time complexity of a single iteration with value $n = s$. Every iteration, we determine every possible next departure time $\tilde{\mathbf{x}}(k+1)$ by applying all matrices in $\mathcal{A}$. This means the time complexity of an iteration is:

$$N(s) = \mathcal{O}(\#\mathcal{A})$$

This time complexity is the same for every iteration and there are m iterations, so

$$TC(GM) = \mathcal{O}(m \times \#\mathcal{A})$$

Thus this method does not have an exponential time complexity, but instead a time complexity that is the product of the resolution time and the amount of adjacency matrices.

We now consider the p-composite greedy delay resolution method. Let m be the resolution time of a delay using the p-composite method. Let $N(s,t)$ be the time complexity of a single iteration spanning t time steps. This means that in an iteration where the solution is not found $t = p$, if a solution is found within the first q time steps of an iteration, then $t = q$. During an iteration, all combinations of index sequences of length t are computed, this gives the time complexity

$$N(s,t) = \mathcal{O}((\#\mathcal{A})^t)$$

The resolution time m is equal to the amount of iteration l times the amount of time steps per iteration p. If the solution is found in the last iteration, then the residual amount of time steps q in this last iteration are also added, so $m = l \times p + q$ and the method has the following time complexity:

$$\begin{aligned} TC(CGM) &= \mathcal{O}(N(0,p) + N(1,p) + \dots + N(l,p) + N(l+1,q)) \\ &= \mathcal{O}((\#\mathcal{A})^p + (\#\mathcal{A})^p + \dots + (\#\mathcal{A})^p + (\#\mathcal{A})^q) \\ &= \mathcal{O}(l \times (\#\mathcal{A})^p) \end{aligned}$$

At first glance, it looks like the composite method exhibits the same exponential time complexity as the combinatorial method, but since we can choose p freely, we can easily limit the influence of the exponent.

We now summarise the results found for the 3 methods in table 4.1. The right-most column of this table shows the amount of delays the method would be applicable to.

Method	Time Complexity	Applicability
Combinatorial Method (CM)	$\mathcal{O}((\#\mathcal{A})^n)$	Always
Greedy Delay Resolution Method (GM)	$\mathcal{O}(n \times \#\mathcal{A})$	Sometimes
Composite Greedy Delay Resolution Method (CGM)	$\mathcal{O}(l \times (\#\mathcal{A})^p)$	Often or always, depending on p

Table 4.1: The three resolution method introduced in this chapter. $n = l \times p + q$ is the resolution time and $\mathcal{A}$ is the adjacency class

To summarise, the combinatorial method can be applied to any delay and can always find a solution to the minimal delay problem if it exists. Because the combinatorial method has a very high computation time for delays that propagate longer, the composite greedy method can be used to find a resolution by trying different values for p. Solutions found by the composite greedy method are not guaranteed to be minimal and if the method does not find a solution for small values of p, it is prone to encountering the same computation time problems as the combinatorial method. Since both the previously discussed methods are not easy to intuitively approach, the regular greedy method can be used when the composite method can not find a solution. In this case, we can manually analyse the delayed departure sequence induced by the method and attempt to find where the problem lies.

4.3. Calamity management: Decoupling

As we have seen in example 7, the delay of one train can spread to other trains in the network. This means that if one train is severely delayed, the entire networks ability to function could come crumbling down. The reason this happens is that even if just one train is delayed, other trains inbound for the same station have to wait for the delayed train to arrive before departing again. It may be in the network's best interest to let these other trains leave the station without waiting for the delayed train. We call this process decoupling, as we remove the (transfer) connection between the delayed train and the other trains at the station.

In this section, we will introduce and discuss decoupling for simple, severe delays. We will also discuss criteria for when decoupling may be used and conclude by introducing a reason for structural decoupling. In this report, we will not go into great details regarding decoupling, in order to avoid its complexity. Some further considerations concerning decoupling can be found in appendix B, but the contents of this appendix will not be used further in this report.

4.3.1. Severe Delays

We illustrate the need for decoupling with an example of a severe delay.

Example 15 *Consider the switching max-plus model $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ with adjacency matrices, initial departure and index sequence respectively*

$$A_0 = \begin{pmatrix} 5 & 5 \\ 5 & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} 3 & 3 \\ 3 & \varepsilon \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 2 \\ 2 & \varepsilon \end{pmatrix}$$

$$\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad J = 0, 0, 0, 0, \ldots$$

This model corresponds to the communication graph with standard commute times:

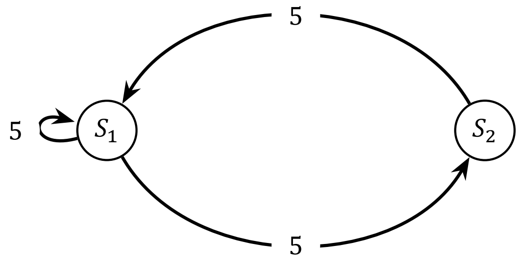


Figure 4.2: The communication graph of the example train network

The departure sequence induced by $\mathbf{x}_0$ *and* J *is:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 5 \end{pmatrix} \rightarrow \begin{pmatrix} 10 \\ 10 \end{pmatrix} \rightarrow \begin{pmatrix} 15 \\ 15 \end{pmatrix} \rightarrow \begin{pmatrix} 20 \\ 20 \end{pmatrix} \rightarrow \begin{pmatrix} 25 \\ 25 \end{pmatrix} \rightarrow \dots$$

Now suppose that one of the trains has stranded for 30 time units, which causes the delay $\delta = (30, 0)^T$*. Resolving this delay using the combinatorial method would yield:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 30 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 32 \\ 32 \end{pmatrix} \rightarrow \begin{pmatrix} 34 \\ 34 \end{pmatrix} \rightarrow \begin{pmatrix} 36 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} 38 \\ 38 \end{pmatrix} \rightarrow \begin{pmatrix} 40 \\ 40 \end{pmatrix} \rightarrow \dots$$

As you can see, it takes many time steps before either train can depart on time. This is due to the fact that in time step $k = 1$*, the train that has just arrived on time at station* S_1 *has to wait for the train on the left-most loop which is running very late. To resolve this issue, we decouple the trains, allowing the train arriving on time to also depart on time:*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 30 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 32 \end{pmatrix} \rightarrow \begin{pmatrix} 34 \\ 10 \end{pmatrix} \rightarrow \begin{pmatrix} 15 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} 38 \\ 20 \end{pmatrix} \rightarrow \begin{pmatrix} 25 \\ 40 \end{pmatrix} \rightarrow \dots \tag{4.2}$$

We see that the resolution time of the delay has not changed, but the value $||\tilde{\mathbf{x}}(k) - \mathbf{x}(k)||_1$ *has been harshly reduced.*

Based on the above example, we incorporate decoupling into the model. We do this by incorporating the decoupling of the connections into the adjacency matrices. The matrix

$$A = \begin{pmatrix} \varepsilon & 2 \\ 2 & \varepsilon \end{pmatrix}$$

is the matrix corresponding to the last delay sequence. This way of modelling decoupling nicely matches the context of switching, as it simply adds new adjacency matrices.

Definition 15 *Let* $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ *be a max-plus switching model. If the communication graphs* $G(A_i)$ *of the adjacency matrices in* $\mathcal{A}$ *do not all have the same arcs, then* $\mathcal{M}_S$ *is called a decoupled max-plus model. Let* $\mathcal{A}_0$ *be the set of all matrices that have the same arcs as* A_0*, we call this set the standard adjacency class and* $\mathcal{A}_\varepsilon = \mathcal{A} \setminus \mathcal{A}_0$ *the decoupled adjacency class.*

Note that in addition to removing arcs, decoupled models can also add arcs by changing an ε in an adjacency matrix to a real number. We can also split the decoupled adjacency class $\mathcal{A}_\varepsilon$ into a partition where each adjacency matrix in the same member of the partition has the same arcs.

Definition 16 *We define* $\mathcal{E}^{(i,j)}$ *to be the matrix with all entries equal to* 0 *except for entry* (i, j) *which is equal to* ε*. We call this matrix the* (i, j)*-decoupling matrix.*

By adding an (i,j)-decoupling matrix to an adjacency matrix A in the plus-times sense $A + \mathcal{E}^{(i,j)}$, we disconnect the train inbound for station i from station j from all other trains outbound from station i.

Example 16 *Consider the matrices*

$$A = \begin{pmatrix} 2 & 2 \\ 2 & \varepsilon \end{pmatrix} \qquad B = \begin{pmatrix} \varepsilon & 2 \\ 2 & \varepsilon \end{pmatrix}$$

B is the $(0,0)$-decoupled matrix of A, as is apparent from the fact that $B = A + \mathcal{E}^{(0,0)}$. If the vertex in $G(A)$ corresponding to index i is labelled S_i, then we can also call B (S_0, S_0)-decoupled for the sake of convenience.

4.3.2. Decoupling Conditions

When looking at equation 4.2 in example 15, we see that after just one iteration with decoupling, the value $||\tilde{\mathbf{x}}(1) - \mathbf{x}(1)||_1$ is immediately harshly reduced. This means that if we gave the greedy method access to decoupled matrices, they would immediately apply it, even for smaller delays. This is not desirable as decoupling comes with a massive practical drawback, namely that passengers can not transfer between decoupled trains. For regular commuters, a regular occurrence of decoupling could therefore be a reason to abandon train commutes. Since we do not want this to happen, we want to limit the use of decoupling only to cases where it is either strictly necessary to resolve delays or vastly beneficial to the delayed departure sequence.

To ensure that decoupling is only used when necessary, we formulate decoupling conditions. These are conditions used to tell the resolution methods when they are allowed to use decoupling. We formulate 3 examples of such decoupling criteria.

- '*If a delay can not be resolved without decoupling, then the use of decoupling is allowed.*'
- '*If switching with decoupling resolves a delay faster than without decoupling, then the use of decoupling is allowed.*'
- '*If switching with decoupling reduces the total delay $||\tilde{\mathbf{x}}(k) - \mathbf{x}(k)||_1$ beyond a certain threshold, then the use of decoupling is allowed.*'

Judging whether these criteria are to be used for a given network, has to be decided on a case-by-case basis. The first criterion may seem useful for each network, but may complicate delay resolution. If resolving the delay as fast as possible is more important in a case than the ability for passengers to make their connections, then the second condition can be used. If reducing travel times for commuters that do not need to transfer is more important in a case than reducing travel times for commuters that do, then the third condition can be used.

4.3.3. Structural Decoupling

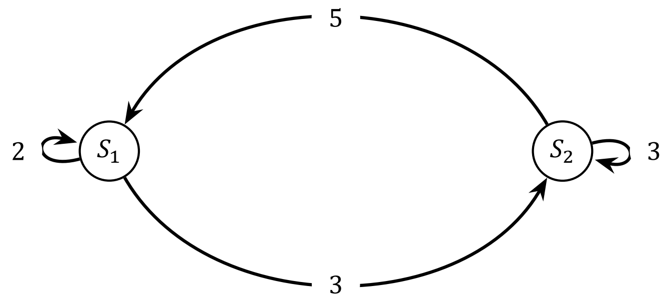
In addition to being useful for managing delays before they are resolved, decoupling can also be used structurally by adding or removing connections during specific times of the day. The amount of trains in the network is determined by the amount of arcs, since every arc has one train. An example of structural decoupling is then to add stations and arcs such that there are enough trains in the network to account for rush-hour. During normal hours, we can then decouple all excess arcs, leaving enough trains for a smaller amount of passengers. In case of calamities, we can also use a different type of decoupling to add arcs. This way, we can create new connections in case existing ones fail. This can be visualised as train companies allowing passengers to make their commute by substitute buses when train lines fail.

Throughout this report, we talk at length about methods for resolving delays. In this endeavour, decoupling is a powerful but precarious tool with many significant benefits and detriments. Because of its complexity, we will not expand this topic too much as to not stray from our initial goal of solving delays. In appendix B, some more ideas surrounding decoupling are discussed. These ideas will not be used for the remainder of this report, as they present some issues that are outside the scope of this report. The ideas surrounding decoupling discussed in this section will be used throughout this report and will prove to be a significant tool for resolving delays.

4.3.4. Early Departures

The max-plus model allows trains to leave their station as soon as all inbound trains have arrived. When decoupling occurs for whatever reason, there are less trains to wait for, so it is possible that trains leave before their scheduled departure time.

Example 17 *Consider the max-plus model corresponding to the following communication graph*



and with corresponding departure sequence

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 20 \end{pmatrix} \to \dots$$

Now suppose that during the first time step, the train travelling from S_2 to S_1 gets decoupled due to a track failure. This means that at time 3, all trains inbound for station S_1 have arrived, so the departure sequence becomes

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 3 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 7 \end{pmatrix} \to \begin{pmatrix} 12 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 15 \end{pmatrix} \to \begin{pmatrix} 20 \\ 20 \end{pmatrix} \to \dots$$

So we see that the sequence has become delayed with a negative delay.

In practice, the above delay can easily be resolved by having a train wait so that the departure in a time step match the expected departure. We can however also prevent this delay from ever happening by restricting trains from leaving ahead of their expected departure time in the case of decoupling.

$$\begin{cases} x(k+1) & = A \otimes x(k) \oplus \tau(k+1) \\ x(0) & = x_0 \end{cases}$$

In the above recurrence relation, the state $\tau(k)$ corresponds to the k'th entry of the departure sequence. In the above example, $\tau(1) = (5,4)^T$, so the early train would not be allowed to leave the station until 5, preventing any delay.

Letting trains wait in stations to match expected departure times can also be used to ensure improved performance of delay resolution method. The reason why we have not used it, is that in the model, there is no restriction on the amount of trains in a station, but in reality, this constraint does exist. If delay resolution method let too many trains wait, some stations may get blocked, causing even more delays. In order to prevent this from happening, we forbid the resolution methods from letting trains wait by not giving them access to the above recurrence relation and only use the above relation in the case of decoupling.

4.4. Network Design

We conclude this chapter by discussing the practical implications of network design. The design of the network we used is in essence the design of the (decoupled) switching max-plus model, which is entirely characterised by 3 things: The adjacency class $\mathcal{A}$, the initial departure $\mathbf{x}(0)$ and the index sequence J.

In practice, we should choose $\mathbf{x}(0)$ and J such that the departure sequence is as convenient as possible for regular commuters. If this condition is established, we can also look at trying to minimise delay propagation in the network for example. As for $\mathcal{A}$, its design is two-fold. On the one hand, we want to design the standard adjacency matrices of $\mathcal{A}$ (i.e., the matrices whose index appears in J) in the same way as $\mathbf{x}(0)$ and J, to benefit regular commuters. On the other hand however, we want to add adjacency matrices to $\mathcal{A}$ that allow us to resolve delays as quickly and efficiently as possible. This latter design is what we will be discussing in this section.

In this chapter, we saw on several occasions that the delay resolution methods may not be able to solve certain delays at all, even when the method had access to significant speed-ups, like in example 13. We saw in this example that the delayed departure could catch up and even surpass the departure sequence, but never actually converge. This inability to solve this delay was due to the fact that the resolution method did not have access to ways to slow down trains.

Example 18 *Consider the switching max-plus model with adjacency matrices*

$$A_0 = \begin{pmatrix} \varepsilon & 6 \\ 6 & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} \varepsilon & 3 \\ 3 & \varepsilon \end{pmatrix}$$

With the departure sequence induced by $\mathbf{x}_0 = (0,0)^T$ *and* $J = (0,0,0,...)$ *being*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 6 \\ 6 \end{pmatrix} \to \begin{pmatrix} 12 \\ 12 \end{pmatrix} \to \begin{pmatrix} 18 \\ 18 \end{pmatrix} \to \begin{pmatrix} 24 \\ 24 \end{pmatrix} \to \begin{pmatrix} 30 \\ 30 \end{pmatrix} \to ...$$

Then the delay $\delta = (1,1)^T$ *can not be resolved.*

In practice, this is not realistic. If a train requires 15 minutes to commute between two stations, then it should also be able to do it in for example 18 minutes[3]. Furthermore, since slowing down trains is often easier than speeding up trains, since the latter comes with an upper bound, adding the ability to slow down to the model seems a realistic extension.

Adding the ability to slow down trains works the same as speeding up trains or decoupling them, namely by changing corresponding matrix entries. Again, since most trains can be slowed down and these slow downs can be as large as possible, this adds a significant amount of new adjacency matrices to the model, which increases the likelihood of the resolution method converging.

In addition to the ability for slow-downs, we can also consider combinations of speed-ups. If we can speed 2 trains by 2 time units each for example, then we can likely also speed up just one of them. It is also reasonable to believe that speeding either up by just 1 time unit is also feasible.

Example 19 *Consider the switching max-plus model* $\mathcal{M}_S$ *with adjacency matrices*

$$A_0 = \begin{pmatrix} 2 & 4 \\ 5 & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 2 \\ 2 & \varepsilon \end{pmatrix}$$

Where A_0 *is the standard matrix. This model corresponds to the following network:*

[3]In theory, it is possible that slowing down trains is not feasible, but this would be in very rare situations where the entire train network is heavily saturated with trains. This would likely be the result of poor network design, which we do not assume to be a factor.

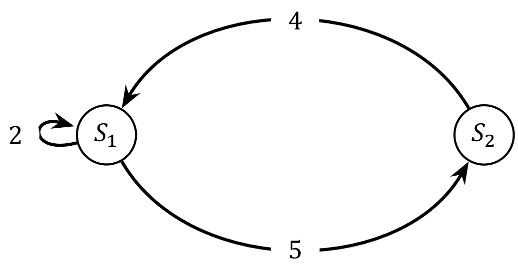


Figure 4.3: The communication graph of the example train network

And matrix A_1 corresponds to slowing down the train from S_1 to S_2 by 3 and the train from S_2 to S_1 by 2. From this, we can expect that combinations of these delays are also possible. Further more, decoupled matrices are likely also feasible. This gives the following adjacency matrices in addition to A_0 and A_1:

$$A_2 = \begin{pmatrix} 2 & 3 \\ 5 & \varepsilon \end{pmatrix} \qquad A_3 = \begin{pmatrix} 2 & 2 \\ 5 & \varepsilon \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 2 & 3 \\ 4 & \varepsilon \end{pmatrix} \qquad A_5 = \begin{pmatrix} 2 & 2 \\ 3 & \varepsilon \end{pmatrix}$$

$$\vdots \qquad \qquad \vdots$$

To save space, we do not write down all matrices, as there are a total of 40.

From this example, it becomes immediately apparent why including each possible adjacency matrix may not be desirable. Even with the improved time complexity of the greedy methods, this vast increase of adjacency matrices proves to be very problematic even for smaller networks. For larger networks, including every combination becomes unmanageable.

We conclude that although we should take decoupled, delayed and combined adjacency matrices into account when resolving delays, they can not be fully incorporated into the practical model without rendering the model unusable due to time restrictions.

5

Multi-Switching Max-Plus

In chapter 3, we introduced the possibility for trains to speed up on certain connections. Part of good network design is ensuring that networks contain some amount of flexibility to ensure that problems occurring on the network can be solved without compromising the logistic capacity of the network. It is possible however, that certain speed-ups are only possible at certain moments in time. There are various reasons why speed ups can only occur under various conditions, such as the following:

- The speedup causes to much noise in residential areas, so trains can not speed up early in the morning or late at night.
- Other, less regular trains such as freight trains or international trains are also using certain connections and overtaking them is not possible.
- Other trains in the system are already using the connections and overtaking them is not possible.

This last reason proves to be rather difficult and as such, we will not take this possibility into account. The issue is briefly discussed in the intermezzo following this chapter. The ability for speed-ups to become unavailable fall under an extension of the switching max-plus model which we will call multi-switching.

In this chapter, we will introduce multi-switching max-plus systems with an example. We will then proceed to formally define the multi-switching max-plus model. When we have established these new concepts, we will differentiate between several types of multi-switching models. Since the model has become more complicated, we will introduce the concept of departure scores to help us resolve delays. We will conclude by modifying the switching based resolution methods to incorporate the previously mentioned scores.

5.1. Freight Train Obstruction

We introduce multi-switching systems using the following example:

Example 20 *We once again consider the example network in section 2.1, as seen in figure 5.1.*

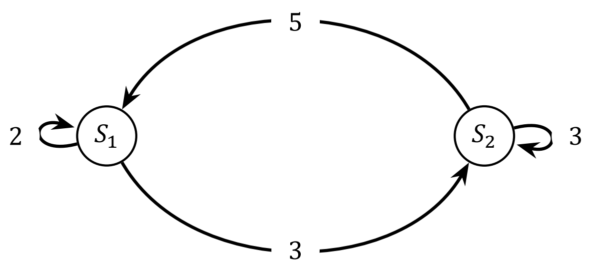


Figure 5.1: The communication graph of the example train network

Both arcs leaving station S_2 can be sped up by 1 time unit.

$$A_0 = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix}$$

and the network has departure the departure sequence induced by $J = 0, 0, 0, \ldots$

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 8 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 11 \end{pmatrix} \to \begin{pmatrix} 15 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 19 \end{pmatrix} \to \ldots$$

We introduce a delay with initial delayed departure $\tilde{\mathbf{x}}(0) = (3, 0)^T$. Using the combinatorial method, we find a possible resolution for this delay is as follows:

$$\begin{pmatrix} S_1 \\ S_2 \\ \tilde{J} \end{pmatrix} : \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 6 \\ 3 \end{pmatrix} \to \begin{pmatrix} 10 \\ 8 \\ 0 \end{pmatrix} \to \begin{pmatrix} 13 \\ 13 \\ 1 \end{pmatrix} \to \begin{pmatrix} 17 \\ 16 \\ 0 \end{pmatrix} \to \ldots$$

So the resolution sequence is $R = (0, 3, 0, 1)$. We now introduce a multi-switching aspect to the system. Every third time step, a freight train uses the upper arc in the inner cycle, making speed ups impossible. This means that during these time steps, we can not use matrices A_1 and A_3, leaving only A_0 and A_2. We now copy these latter two matrices $B_0 = A_0$ and $B_2 = A_2$ and put them in a new adjacency class $\mathcal{B} = \{B_0, B_2\}$. The sequence for which adjacency matrix is available in each time step is now $\mathcal{H} = (\mathcal{B}, \mathcal{A}, \mathcal{A}, \mathcal{B}, \mathcal{A}, \mathcal{A}, \ldots)$. When looking at the resolution sequence, we see that $R(3) = 1$, but $\mathcal{H}(3) = \mathcal{B}$, so adjacency matrix A_1 is not available during this time step. We thus need to only take the available adjacency matrices into account for the combinatorial method. When doing this, we find the following equality

$$\mathbf{x}(6) = A_1 \otimes A_0 \otimes B_0 \otimes A_0 \otimes A_3 \otimes B_0 \otimes \tilde{\mathbf{x}}(0),$$

so the resolution sequence in the multi-switching system is equal to $R = (0, 3, 0, 0, 0, 1)$.

We see that the multi-switching component changed the resolution time from 4 to 6. This shows us that adding the multi-switching component to the model can drastically influence delay resolution. The change from a switching model to a multi-switching model in this situation, was due to the addition of the freight train. This freight train is a train that is not modelled by the network, but does occasionally use the connections in it. The following are more general examples of situations that warrant multi-switching:

- **External trains:** Trains outside the network using connections.
- **Speed limits:** During certain times of day, like late at night or during rush-hour, speeding up trains may cause excess noise or risk.
- **Delay:** Delays can also be dynamically modelled using multi-switching. This will be discussed in section 5.3.

An important observation we make based on the above situations, is that we have no control over the multi-switching aspects of a system. For the index sequence which decides which matrix in a class to use, we can choose which trains to speed up or slow down, but the sequence choosing which adjacency class can be used in each time step, is determined entirely by external factors outside of our control.

Definition 17 *Let Ω be a set of adjacency classes and let $\mathbf{x}_0 \in \mathbb{R}^n_{max}$. We call $\mathcal{M}_M = \mathcal{M}_M(\Omega, \mathbf{x}_0)$ the multi-switching max-plus model of Ω with initial departure $\mathbf{x}_0$ when*

$$\mathcal{M}_M : \begin{cases} \mathbf{x}(k+1) & = A(k) \otimes \mathbf{x}(k) \\ \mathbf{x}(0) & = \mathbf{x}_0 \end{cases}$$

Where $A(k) \in \mathcal{A}(k)$ is the adjacency matrix applied in time step k and $\mathcal{A}(k)$ is the adjacency class available in time step k. We call Ω the adjacency array of $\mathcal{M}_M$

This definition is identical to the definition of the switching max-plus model (definition 8) apart from the last part. As such, the choice function and index sequence J have essentially the same meaning. The aspect that is added to this definition, is the sequence that determines which adjacency class is applied in which time step. For this, we use another index sequence which we call the class sequence.

Definition 18 *Let $\mathcal{M}_M$ be a multi-switching max-plus model with adjacency array $\Omega = \{\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_m\}$. Let $\mathcal{H}$ be a sequence of indices $H_i \in \{0, \ldots m\}$. If $H_k = i$ (also denoted $\mathcal{H}(k) = i$), then*

$$\mathbf{x}(k+1) = A(k) \otimes \mathbf{x}(k)$$

where $A(k) \in \mathcal{A}_i$. We call this $\mathcal{H}$ the class sequence of $\mathcal{M}_M$

If the class sequence is known, we can also denote $\mathcal{M}_M = \mathcal{M}_M(\Omega, \mathbf{x}_0, \mathcal{H})$ and if even J is known then $\mathcal{M}_M = \mathcal{M}_M(\Omega, \mathbf{x}_0, \mathcal{H}, J)$. In this latter case, the model will produce a single departure sequence. We illustrate these definitions by applying them to example 20

Continuation of example 20 *In this example, we have that $\Omega = \{\mathcal{A}, \mathcal{B}\}$ where*

$$\mathcal{A} = \left\{ \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}, \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix}, \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix}, \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix} \right\}$$
$$\mathcal{B} = \left\{ \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}, \qquad \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix} \qquad \right\}$$

labelled $\mathcal{A} = \{A_0, A_1, A_2, A_3\}$ and $\mathcal{B} = \{B_0, B_2\}$. We have that the initial departure is $\mathbf{x}(0) = (1, 0)^T$, the class sequence is $\mathcal{H} = (1, 0, 0, 1, 0, 0, \ldots)$ and the index sequence is $J = (0, 0, 0, \ldots)$. Notice that since $A_0 = B_0$, the class sequence has no influence on the matrix being chosen by this index sequence. We want to find the resolution sequence R so that R resolves the delay with initial delayed departure $\tilde{\mathbf{x}}(0) = (3, 0)^T$. As seen in the example, the solution to this problem is $R = (0, 3, 0, 0, 0, 1)$.

Like for switching, we define some additional concepts for multi-switching

Definition 19 *Let $\Omega = \{\mathcal{A}_0, \ldots, \mathcal{A}_n\}$ be an adjacency array. We call $\mathcal{A}_0$ the standard adjacency class of Ω. Let $\mathcal{H}$ be a class sequence and $\mathcal{H}(k) = i$, then $\mathcal{A}_i$ is called the expected adjacency class of time step k. If $\mathcal{A}_j, j \neq i$ is used in time step k, we say that the network has shifted from $\mathcal{A}_i$ to $\mathcal{A}_j$.*

Now that we have established a basic understanding of multi-switching max-plus models, we translate the delay problem to match this new model.

Problem 4 *The Delay Problem:*
Let $\mathcal{M}_M = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, \mathcal{H}, J)$ be a multi-switching max-plus model. Given a delayed departure $\tilde{\mathbf{x}}(m)$, give an index sequence $\tilde{J}$ such that the delayed departure sequence $(\tilde{\mathbf{x}}(k))_{k \geq m}$ induced by $\tilde{\mathbf{x}}(m)$ and $\tilde{J}$ converges to the departure sequence of $\mathcal{M}_M$ as quickly as possible, i.e. with the smallest possible resolution time.

To give an intuitive interpretation of how delay resolution works, we give the order of events for multi-switching systems in every time step.

1. We start with the class index $\mathcal{H}(k) = 0$ for the standard adjacency class.
2. We evaluate the current situation to determine the shift of the network in the given situation $\mathcal{H}(k) = i$.
3. From adjacency class $\mathcal{A}_i$, we choose the most suitable matrix $A_j \in \mathcal{A}_i$.
4. We apply A_j to the current departure.

Which corresponds to the following scheme:

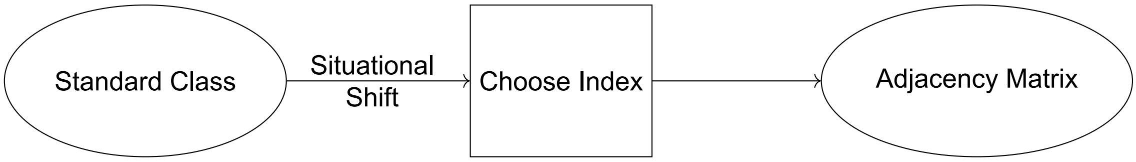


Figure 5.2: The order of event in a multi-switching system

In this scheme, the square is the only moment where we can influence the system by making a decision.

5.2. Departure Score

The addition of multi-switching makes the model more realistic, as we can now handle external factors influencing the model. The problem that arises however, is that as we have seen in example 20, resolving delays can become more difficult. Furthermore, the tools available to the system for solving a delay can change at every time step, making delay resolutions less direct.

To combat the issue of more complicated delay resolutions, we give each delayed departure a score to reflect how good it is compared to the expected departure. The problem of resolving the delay over an unknown amount of time steps then becomes equivalent to minimising the score over a given amount of time steps, simplifying the problem.

Definition 20 *Let $X = (\mathbf{x}(k))_{k\geq 0}, Y = (\mathbf{y}(k))_{k\geq 0}$ be departure sequences with the same dimension. We call $\mathbf{s} = \mathbf{s}(X, Y, k) \in \mathbb{R}^n_{\geq 0}$ a score vector of $\mathbf{x}(k)$ in X with respect to Y if $\mathbf{s} = 0$ if and only if $\mathbf{x}(k) = \mathbf{y}(k)$.*

Based on this concept of score, we can now design algorithms to resolve delays. The way we do this is to at each time step choose the course of action that leads to the 'minimal' score in the hopes of at some point reaching the score 0, as this results in delay being resolved by the definition of scores. Before we can minimize scores however, we first need to order them.

Definition 21 *Let $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{R}^n_{\geq 0}$ be departure scores. We define the score ordering as follows: $\mathbf{s}_1 < \mathbf{s}_2$ if the first non-zero component of $\mathbf{s}_1 - \mathbf{s}_2$ is negative. If all components are zero then $\mathbf{s}_1 = \mathbf{s}_2$.*

We notice that in the greedy methods (method 4 and method 5), we wanted to minimize 2 metrics. First we wanted to minimize the metric $s_1 = ||\tilde{\mathbf{x}}(k) - \mathbf{x}(k)||_{max}$ and then we wanted to minimize the metric $s_2 = ||\tilde{\mathbf{x}}(k) - \mathbf{x}(k)||_1$. If we define a score based on these metrics we get:

$$\mathbf{s}(X, Y, k) = \mathbf{s}(\tilde{\mathbf{x}}(k), \mathbf{x}(k)) = \begin{pmatrix} ||\tilde{\mathbf{x}}(k) - \mathbf{x}(k)||_{max} \\ ||\tilde{\mathbf{x}}(k) - \mathbf{x}(k)||_1 \end{pmatrix}$$

In the following example, we will see that minimizing the 2 metrics as done by the greedy methods is the same as minimizing the score vector.

Example 21 *Consider three possible delayed departures which, when compared to the expected departure, give the following metrics*

$$||\tilde{\mathbf{x}}_1 - \mathbf{x}||_{max} = 1 \qquad ||\tilde{\mathbf{x}}_1 - \mathbf{x}||_1 = 7$$
$$||\tilde{\mathbf{x}}_2 - \mathbf{x}||_{max} = 1 \qquad ||\tilde{\mathbf{x}}_2 - \mathbf{x}||_1 = 6$$
$$||\tilde{\mathbf{x}}_3 - \mathbf{x}||_{max} = 2 \qquad ||\tilde{\mathbf{x}}_3 - \mathbf{x}||_1 = 2$$

The greedy methods will first choose $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ based on their minimal max-norm. From these two, $\mathbf{x}_2$ will then be chosen based on its minimal 1-norm. We now look at the score of each departure where $\mathbf{s}(\tilde{X}, X, k) = (||\tilde{\mathbf{x}} - \mathbf{x}||_{max}, ||\tilde{\mathbf{x}} - \mathbf{x}||_1)^T$:

$$\mathbf{s}_1 = \begin{pmatrix} 1 \\ 7 \end{pmatrix} \qquad \mathbf{s}_2 = \begin{pmatrix} 1 \\ 6 \end{pmatrix} \qquad \mathbf{s}_3 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

And we look at their pairwise differences $\mathbf{d}_{ij} = \mathbf{s}_i - \mathbf{s}_j$:

$$\mathbf{d}_{12} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \mathbf{d}_{13} = \begin{pmatrix} -1 \\ 5 \end{pmatrix} \qquad \mathbf{d}_{23} = \begin{pmatrix} -1 \\ 4 \end{pmatrix}$$

From this, we can derive the inequalities

$$s_2 < s_1 < s_3$$

We see that just like for the greedy methods, $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ are more favourable than $\tilde{\mathbf{x}}_3$ and $\tilde{\mathbf{x}}_2$ is more favourable than $\tilde{\mathbf{x}}_1$.

Since minimising the score is the same as minimising the metric criteria, we can reformulate the greedy method.

Method 6 *p-Composite Greedy Delay Resolution Method*
Let $\mathcal{M}_M = \mathcal{M}_M(\Omega, \mathbf{x}_0, \mathcal{H}, J)$ be a multi-switching max-plus model. Let $\tilde{\mathbf{x}}(m)$ be a delayed departure. p is a natural number.

1. *Let $n = 0$*
2. *Use the combinatorial method to determine a finite sequence $\tilde{R}_n$ of length p. such that:*

$$\tilde{\mathbf{x}}(m + (n+1) \times p) = \bigotimes_{i=1}^{p} \left(A_{\mathcal{H}(m+n\times p), \tilde{R}_n(p-i)}\right) \otimes \tilde{\mathbf{x}}(m + n \times p)$$

   Where $\mathbf{s}(\tilde{X}, X, m+n+1)$ is minimised.
3. *If $\exists q \leq p : \tilde{\mathbf{x}}(m + n \times p + q) = \mathbf{x}(m + n \times p + q)$, shorten $\tilde{R}_n$ to its first q entries. We are done.*
4. *Choose one of the resulting finite sequences $\tilde{R}_n$ based on some criteria.*
5. *Increment n by 1 and return to step 2*

Where $A_{\mathcal{H}(m+n\times p), \tilde{R}_n(p-i)}$ is the adjacency matrix corresponding with the index $\tilde{R}_n(p-i)$, in the adjacency class corresponding with index $\mathcal{H}(m + n \times p)$. $n \times p + q$ is the resolution time and the concatenation of all $\tilde{R}_i$ is the resolution sequence.

With the regular greedy method still being the same as the 1-composite greedy method. This more general formulation of the greedy method also allows us to change the score function to possibly find a more favourable delay resolution criterion. We also notice that the criteria mentioned in step 4 was previously used to find minimal solutions, which could also be incorporated into the score. Not doing this however, grants us the possibility to differentiate between the criteria that resolve a delay efficiently and the criteria that choose a suitable resolution from a set of possible efficient resolutions.

5.3. Modelling Delays

With the addition of multi-switching, we can now also dynamically model delay. Up to this point, we have only considered single delays and the delayed departure sequence they induced. In reality however, delays can happen at any time and if the resolution methods are to be used in practice, the delays need to be modelled in a manner that is as realistic as possible. They way delays have been modelled so far is as follows:

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k) + \delta(k)$$

Where $\delta(k)$ signifies the delay. With multi-switching we can model this delay in a more natural way whilst also accounting for the fact that there are different types of delay. The notions of these different types will come in handy in chapter 6 where we will make a simulation based on multi-switching delay modelling.

In this section, we will model delays into a multi-switching model using delayed adjacency classes. We will then look at how delayed classes affect the resolution methods by defining the concepts of anterior and posterior indexing. We will conclude by showing how stochastic multi-switching delay modelling can be applied in practical situations.

5.3.1. Delayed Classes

The idea of delayed adjacency classes, which we will also call delayed classes, is that every adjacency matrix is linked to a delayed adjacency matrix via some delay. We illustrate this with an example.

Example 22 *Consider the following communication graph*

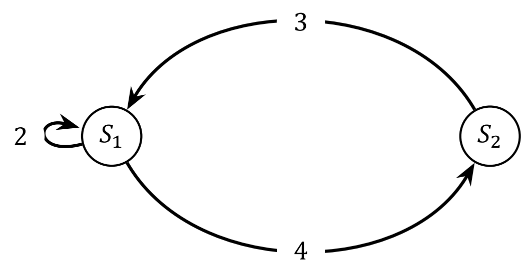


Figure 5.3: The communication graph of the example train network

With adjacency matrices

$$A_0 = \begin{pmatrix} 2 & 3 \\ 5 & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} 2 & 3 \\ 4 & \varepsilon \end{pmatrix}$$

and departure sequence

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 4 \\ 5 \end{pmatrix} \to \begin{pmatrix} 8 \\ 9 \end{pmatrix} \to \begin{pmatrix} 12 \\ 13 \end{pmatrix} \to \begin{pmatrix} 16 \\ 17 \end{pmatrix} \to \dots$$

Now suppose that the train on the bottom arc of the central cycle is delayed by 1 time unit. This means we need to add one to the matrix entries corresponding to that commute

$$\tilde{A}_0 = \begin{pmatrix} 2 & 3 \\ 6 & \varepsilon \end{pmatrix} \qquad \tilde{A}_1 = \begin{pmatrix} 2 & 3 \\ 5 & \varepsilon \end{pmatrix}.$$

This results in the delayed departure sequence:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 4 \\ 6 \end{pmatrix} \to \begin{pmatrix} 9 \\ 9 \end{pmatrix} \to \begin{pmatrix} 12 \\ 14 \end{pmatrix} \to \begin{pmatrix} 17 \\ 17 \end{pmatrix} \to \dots$$

As we can see, this has the same effect as adding the delay $\delta(0) = (0,1)^T$. We now suppose that the left-most arc is delayed by 1 time unit. This yields the following delayed adjacency matrices:

$$\tilde{A}_0 = \begin{pmatrix} 3 & 3 \\ 5 & \varepsilon \end{pmatrix} \qquad \tilde{A}_1 = \begin{pmatrix} 3 & 3 \\ 4 & \varepsilon \end{pmatrix}$$

and departure sequence

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \begin{pmatrix} 4 \\ 5 \end{pmatrix} \to \begin{pmatrix} 8 \\ 9 \end{pmatrix} \to \begin{pmatrix} 12 \\ 13 \end{pmatrix} \to \begin{pmatrix} 16 \\ 17 \end{pmatrix} \to \dots$$

This means that even though there is a delay, it does not propagate.

The above example illustrates that depending on where the delay happens, it may propagate more, less or even not at all. Determining this without the use of delayed classes would require calculations to be done outside of the model. By letting delays change the adjacency matrices through multi-switching, no additional calculations have to be done. As we can also see by the example, all the delayed matrices are transformations of their original counterparts. We can place these delayed matrices in a new adjacency class which we will call the delayed class

Definition 22 *For any $c \in \mathbb{R}_{max}$, we define $c^{(i,j)}(m,n)$ as the $m \times n$ matrix with all elements equal to 0 except for element (i,j), which is equal to c. If the size of the matrix is clear from the context, then we write $c^{(i,j)}$.*

Definition 23 *Let $\mathcal{A}$ be an adjacency class and let $\tilde{\mathcal{A}}^{(i,j)} = \{A + c^{(j,i)} : A \in \mathcal{A}\}$. Then $\tilde{\mathcal{A}}^{(i,j)}$ is called a (i,j)-delayed adjacency class of $\mathcal{A}$ with delay c.*

If several delays occur in a single time step, we simply call the resulting adjacency class the delayed adjacency class of that time step.

5.3.2. Anterior and Posterior Indexing

Because delays are often not predictable, we can not base the decisions in a time step on the delays that will happen during that time step. This means the matrix we really apply may differ from the matrix that was originally chosen.

Example 23 *Consider the multi-switching max-plus model with $\Omega = \{\mathcal{A}, (\bigcup \tilde{\mathcal{A}})\}$, where $\bigcup \tilde{\mathcal{A}}$ is the collection of every possible delayed adjacency class and where*

$$\mathcal{A} = \left\{ \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}, \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix}, \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix}, \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix} \right\}$$

and with departure sequence and delay respectively

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 16 \end{pmatrix} \to \dots$$

$$\delta(1) = \begin{pmatrix} 0 \\ 2 \end{pmatrix}.$$

When applying the greedy algorithm for 1 time step, it chooses the second matrix in $\mathcal{A}$*,* A_1*. This would result in* $x(1) = (6,4)^T$*. However, due to a delay, a delayed adjacency matrix is applied instead:*

$$A_1 = \begin{pmatrix} 2 & 4 \\ 3 & 3 \end{pmatrix} \qquad \rightarrow \qquad \tilde{A}_1 = \begin{pmatrix} 6 & 4 \\ 3 & 3 \end{pmatrix}$$

This results in $x(1) = (7,4)^T$*, which does not match the expected result. Since the delay could not be foreseen, the algorithm can not influence this altered outcome.*

When delays are the only cause for shifting, shifts only happen after an index has already be chosen. This order of event corresponds with the following scheme:

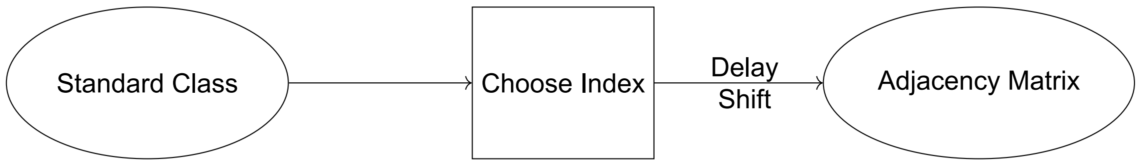


Figure 5.4: The order of event in a multi-switching system with anterior indexing

Since the index is chosen before the shifting has occurred, we call this anterior indexing. An important property of anterior indexing is that every adjacency class must be of the same size. The reason for this is that since the index i is chosen based on the standard adjacency class $\mathcal{A}_0$, the index i must also make sense for the other adjacency classes. We notice that since the delay shift happens after choosing the index, it does not have any influence on the choice, only on the effect.

The opposite of anterior indexing is posterior indexing. This happens when the index is chosen after the shifting has occurred. In this case, the choice is based on all information in the time step as no further shifting will occur after the choice is made. It is also not necessary for the adjacency classes to be of the same size as the index is chosen based on the adjacency class that was shifted to.

In many cases, shifting can occur both before and after indexing. We call this mixed indexing and it corresponds to the following order of events scheme:

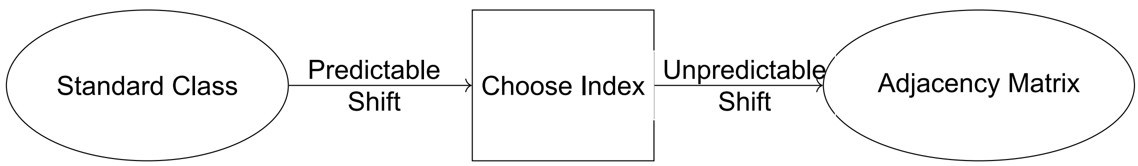


Figure 5.5: The order of event in a multi-switching system with mixed indexing

We choose the index of the time step in the same way as normally, basing the decision on the known information. The adjacency matrix corresponding to this index when a delay occurs is then exactly the delayed adjacency matrix of the originally chosen matrix.

5.3.3. Delays in Simulations

Now that we have added delays to the model, we can introduce delays to a departure sequence dynamically. This means we are no longer limited to having just one predetermined set of delays. By adding delays at random to a network, we can simulate the course of a real train network, which we will do in chapter 6. Dynamic delays are not useful for formulating intuitive delay resolution methods, as their random nature would make this too complicated, but they can be used to evaluate the performance of resolution methods in more realistic systems. For now, we will limit ourselves to discussing how we can model dynamic delays in a realistic manner.

In this report, we will consider 3 possible types of delays. The occurrence and sustaining of these delays is randomly distributed using Bernoulli distributions for simplicity, a delay either occurs, or it does not. The types we consider are:

- **Track Delays:** Due to rail or train obstructions, the weight of one arc is increased by a constant.
- **Station Delays:** Due to a departure problem, all trains at a station leave at a later time.
- **Track Failure:** Due to a catastrophic rail or track incident, one arc can not be used.

As previously said, in simulation, the onset of any of these delays happens according to a Bernoulli distribution. For track delays and failures however, it is possible that the delay is not resolved in one time step. In these cases, their sustaining is also modelled according to a Bernoulli distribution. We assume that station delays to not sustain like this. We first look at the onset distributions:

$$\begin{aligned} &\mathbb{P}(O(TD)) = p_1 && O(TD) = \text{Onset Track Delay}\\ &\mathbb{P}(O(SD)) = p_2 && O(SD) = \text{Onset Station Delay}\\ &\mathbb{P}(O(TF)) = p_3 && O(TF) = \text{Onset Track Failure}\end{aligned}$$

where p_1, p_2 and p_3 are probabilities between 0 and 1 and the onset variables are all binary variables. Now considering the resolution:

$$\begin{aligned} &\mathbb{P}(S(TD)) = q_1 \times TD && S(TD) = \text{Sustain Track Delay}\\ &\mathbb{P}(S(SD)) = 0 && S(SD) = \text{Sustain Station Delay}\\ &\mathbb{P}(S(TF)) = q_3 \times TF && S(TF) = \text{Sustain Track Failure}\end{aligned}$$

where q_1 and q_3 are probabilities between 0 and 1 and the sustaining variables are binary variables. The variables TD, SD and TF are also binary variables that start at 0, become 1 when the corresponding onset variable is one and returns to 0 when the corresponding sustaining variable is zero. In practice, different types of delays and distributions may be more accurate. When creating a multi-switching max-plus simulation based on a real train network, these delays and distributions will have to be determined empirically and statistically.

5.3.4. Implementation of Adjacency Classes

When implementing simulations for multi-switching max-plus systems with mixed indexing, we can implement every adjacency class that is caused by a predictable shift, as there are generally not that many and they are required to make shifting decisions. We can however not implement every possible delayed adjacency class as theoretically, there is an infinite amount of possible delays. Furthermore, since delays can not be used to make shifting decisions, there is no need for them to be implemented at the beginning of the simulation. Instead, we just apply delays that occur to each of the adjacency classes which yields all delayed adjacency classes. We illustrate this with an example.

Example 24 *Consider a multi-switching max-plus model with adjacency array*

$$\Omega = \{\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2\} \cup \left\{\left(\bigcup \tilde{\mathcal{A}}_0\right), \left(\bigcup \tilde{\mathcal{A}}_1\right), \left(\bigcup \tilde{\mathcal{A}}_2\right)\right\}.$$

The unions of delayed adjacency classes can be infinitely large, but only a finite number of them are used. At every time step, the predictable shift yields one of the adjacency class, call the resulting class $\mathcal{A}_i$. A function signifying potential delays is then applied to $\mathcal{A}_i$. This function projects this adjacency class to itself if there is not delay and to the appropriate delayed class if there is a delay:

$$\delta : \mathcal{A}_i \mapsto \tilde{\mathcal{A}}_i^\delta.$$

As an example, we take the adjacency class

$$\mathcal{A}_i = \left\{\begin{pmatrix} 3 & 3 \\ 4 & 4 \end{pmatrix}, \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}\right\}$$

with the possible delays:

$$\mathcal{A}_i \xrightarrow{TD(0,1):5} \left\{ \begin{pmatrix} 3 & 3 \\ 9 & 4 \end{pmatrix}, \begin{pmatrix} 2 & 2 \\ 8 & 3 \end{pmatrix} \right\}$$

$$\mathcal{A}_i \xrightarrow{SD(1):4} \left\{ \begin{pmatrix} 3 & 7 \\ 4 & 8 \end{pmatrix}, \begin{pmatrix} 2 & 6 \\ 3 & 7 \end{pmatrix} \right\}$$

$$\mathcal{A}_i \xrightarrow{TF(1,0)} \left\{ \begin{pmatrix} 3 & \varepsilon \\ 4 & 4 \end{pmatrix}, \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix} \right\}$$

5.4. The Scoring Problem

We have seen that greedy methods can be generalised using the concept of scores. All the greedy methods do is minimise a score over a certain number of steps. The score function we chose for the examples in this report has the max-norm and the 1-norm as components, but completely different score functions can be chosen. The greedy methods therefore correspond not just with one method, but with an entire class of methods, each utilising different scoring criteria. This gives rise to the score problem for delay resolution.[1]

Problem 5 *Given a delay problem, which score function* $s : \{\tilde{\mathbf{x}}(k)\}_{k\geq 0} \rightarrow \mathbb{R}^n_{\geq 0}$ *yields the most efficient instance of a* p*-composite greedy delay resolution method?*

The solution to this problem may vary based on the network and possible delays. The solution can be attempted to be solved either by closely analysing networks and adjacency matrix interactions to devise an optimal score deductively, or by analysing optimal solutions and extrapolating an optimal score inductively. In the case where a lot of data is available, statistical methods may even be used. Using state scores also opens the door for machine learning algorithms to appoint score, which they can be taught using reinforcement learning.

The scoring problem is a massive problem on its own, and outside the scope of this report. A major downside of using scoring criteria based on non-obvious metrics, is that the resolution of delays becomes a black-box process. The intuition will become so convoluted that if the algorithms were to fail, a make-shift resolution could not be intuitively created using human intervention. Because of the above reasons, we keep using the previously formulated scoring criteria.

[1] Note that the scoring problem can be formulated for any state-based system where we have control over the next state.

Intermezzo: Modelling Restrictions and Systems Theory

Before moving on to simulate a real train network, we will briefly discuss some restrictions of the max-plus models. The max-plus models aim to model logistics networks in a simple and intuitive manner. This simplicity sometimes causes the model not to be realistic for networks in the real world. In this chapter, we will briefly discuss 3 such restrictions: desynchronisation, time offset and recoupling. After this, we will discuss the link between the topics discussed in the previous chapters and mathematical systems theory.

Desynchronisation

When train networks contain a cycle, the trains on the cycle travel continuously to consecutive stations on this cycle. Consider a train commuting on a cycle with 3 stations S_1, S_2, S_3 for example with all commute times equal to 1, as seen in figure 5.6. Depending on the direction in which the train drives, the train will repeatedly travel through these 3 stations in a 3-periodic fashion. On this cycle, there will be

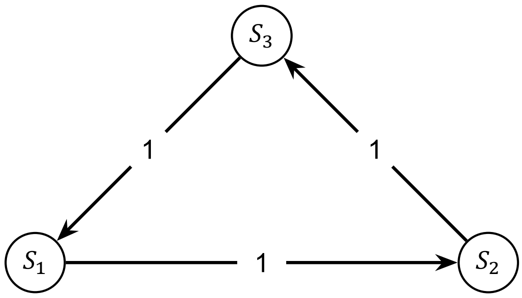


Figure 5.6: Triangular Network

3 trains present at any time, one between each of the stations. We let all trains depart their starting station at time 0 to their next station. The train travelling from S_1 to S_2 incurs a delay of 5 time units due to an engine failure. This means the next departure at station S_2 will be 6. However, the train starting at S_3 has already travelled through S_1 and to S_2 by this time, surpassing the delayed train. This means there is a train at station S_2 ready to depart at time 2.

In the max plus model, all trains are believed to traverse exactly one arc per time step. As we can see above however, it is possible that due to delays, one train can traverse many arcs in the same time that another train can traverse one arc. We will call the property of max-plus systems that trains travel one arc per time unit the synchronicity of max-plus systems. In events like described above, breaking this synchronicity is a very logical step, which we call desynchronisation. Simply decoupling the delayed train does not suffice to desynchronise the network, as this simply means other trains can depart station S_2 while no train from S_1 has arrived. It can be achieved however, by adding an arc from S_3 to S_2 with weight 2. Doing this complicates the max-plus model a lot, as like delays, these ghost routes have to be added dynamically in the case of severe delays. As such, we will not allow for desynchronisation in any simulations.

Time Offset

A problem that can be encountered in multi-switching max-plus models is the occurrence of time offset. Consider a train network like the one in figure 5.6. Suppose that on any time interval of the form $[3k, 3k+1], k \in \mathbb{Z}$, a freight train uses the arc (S_1, S_2) so that no other trains can change their commute time at this time. If there are no delays, this obstacle corresponds to the class sequence $\mathcal{H} = 1, 0, 0, 1, 0, 0, \ldots$ where $\mathcal{A}_1$ corresponds to the impossibility of commute changes on (S_1, S_2). This class sequence is a direct result from the departure sequence:

$$\begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} \rightarrow \begin{pmatrix} 6 \\ 6 \\ 6 \end{pmatrix} \rightarrow \ldots$$

Now suppose that at time step $k = 1$, all trains are delayed by 1 time unit. This changes the departure sequence to

$$\begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} \rightarrow \begin{pmatrix} 6 \\ 6 \\ 6 \end{pmatrix} \rightarrow \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix} \rightarrow \ldots$$

So the class sequence is $\mathcal{H}' = 1, 0, 1, 0, 0, 1, 0, \ldots$. It is plausible to believe that external trains such as freight trains to not restrict themselves to our time steps, but rather to their own timetables. This means that if commute times change, the time steps in which these freight trains obstruct the system is not perfectly predictable, but dependent on random delays. As such, to properly model external trains using the network, we would need to dynamically change the class sequence. Since delays can happen in between time steps, doing so is not always possible, as we would need to change the class index in the middle of the time step it correlates to. In order to avoid further complicating the already extensive multi-switching and in order to prevent improper modelling, we will assume that external trains always take the time steps of the network into account.

Recoupling

The last modelling constraint we will discuss in this chapter concerns decoupled trains. In the max-plus models, by design, decoupled trains are trains that are disconnected from their destination station. This means that for as far as the model is concerned, the train never actually arrived at the station. In reality, this is of course not the case; the train did arrive, but at a later time. In most cases we have seen thus far in this report, trains arriving at a station are the same trains that leave that station in the next time step. This means that the delayed train, despite never having arrived at the station, does leave it again. If no train is present to substitute the decoupled train, this means there is a ghost train in the network. The recoupling problem is concerned with ensuring that this ghost train and its corresponding decoupled train reach the same point in the network as soon as possible.

The recoupling problem is an issue of train logistics and as such, its solutions are dependent on properties of the given train network. Solutions can strongly differ based on the amount of backup trains or safety nets to prevent major errors. These properties generally fall outside of the max-plus models we employ and as such, we will not implement methods for dynamic recoupling. Since the recoupling problem is in essence the same problem that arises for the transition to and from rush hour however, as is discussed in appendix B, we assume that the same method can be used where all trains resume their regular schedule, including the ghost train, and the decoupled train manoeuvres with the other trains to at some point coincide with the ghost train.

Systems Theory

The concepts we have defined and discussed in this report, have been formulated in a manner that is as intuitive as possible. To this end, we did not lean much on existing mathematics for new concepts, so that every step makes sense on its own. There is however a strong link between the concepts discussed in this report an conventional systems and control theory. While we will not confuse the reader by introducing alternative notation for the concepts we have defined, we will discuss this link in order to provide some additional context for max-plus algebra and its extensions. This extra context will be provided in appendix C for the interested reader.

6
Simulating Train Networks

In the previous chapters, we have aimed to build a solid basis for max-plus models and their extensions in order to depict a real train network in a somewhat realistic manner. In this chapter, we will use the acquired models to simulate a real train network for an entire day. The aim is to create a realistic train network where delays occasionally occur and where the methods are used to resolve delays as quickly as possible. We will achieve this result by first creating the simulated train network and establishing a periodic timetable. We will then dynamically model delays as discussed in chapter 5 and use the delay resolution methods to resolve these delays. Applying methods to a randomised simulation is a good way of benchmarking any problem solving method, we will thus formulate some statistics for the simulation that can be used to aid the benchmarking process.

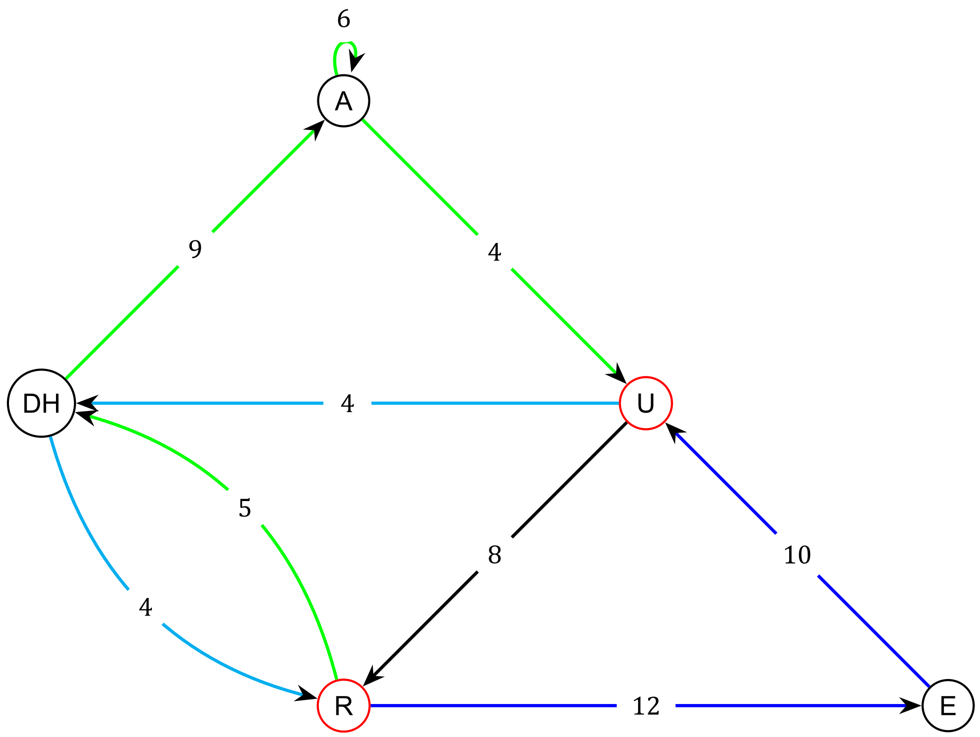


Figure 6.1: The subject network.

6.1. The Simulated Network

The network we will simulate is a small part of the dutch railway network, as can be seen in figure 6.1. The simulation we will construct could be used in the case where the systems currently in use by the network operator fail, in order to make sure that at least a primitive amount of train travel is possible. Since we want the most important part of the train network to be operational, the network will include many of the major dutch cities. In order to give meaning to the weights of the graph, we assume that 1 time unit corresponds to 6 minutes in the real world. The letters at each of the stations correspond with the following cities:

E: Eindhoven
U: Utrecht
A: Amsterdam
DH: The Hague
R: Rotterdam

The colours of the arcs correspond with train routes. Trains travelling on identically coloured arcs need to wait for each other, but train on different coloured arcs do not. This means that the train travelling from DH to R will not wait for the train from R to DH before departing. In order to model this disconnect, we split station DH into 2 separate stations, as can be seen on figure 6.2 On red coloured

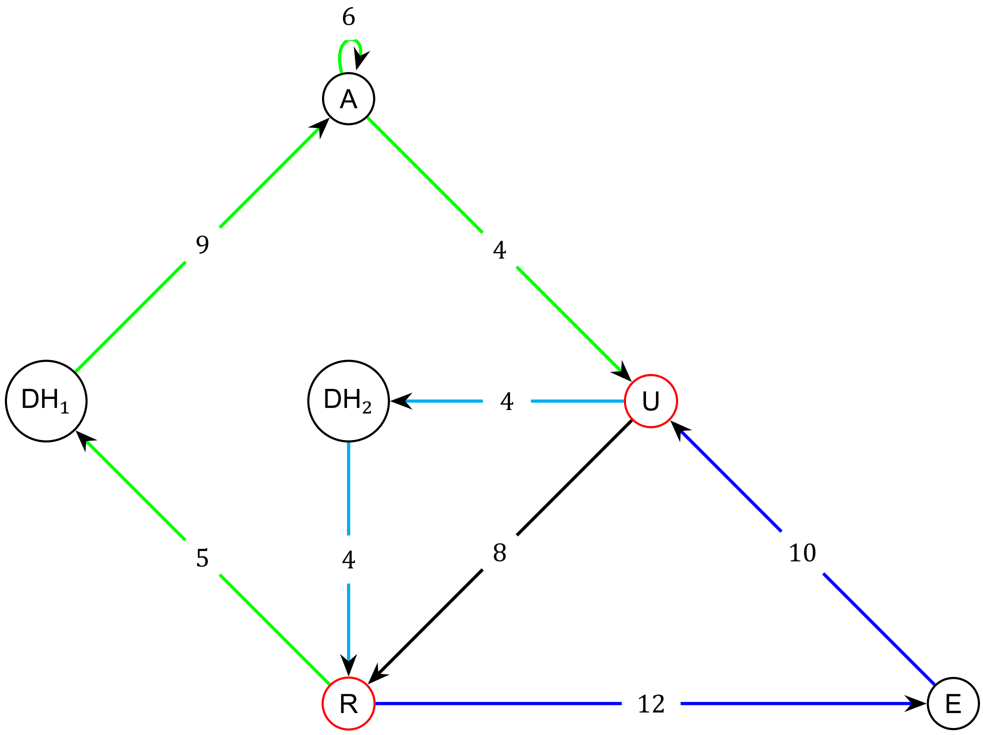


Figure 6.2: The subject network. The station corresponding to The Hague is split into two separate routes.

stations, all trains need to wait for each other. This means that regardless of the colours of the incoming arcs, these stations never need to be split. In this network, we deem Utrecht and Rotterdam to be important stations, so they will allow transfers between all trains.

To construct the standard adjacency matrix, we construct a table with all arcs and their weights.

	E	U	A	DH_1	DH_2	R
E:	ε	ε	ε	ε	ε	12
U:	10	ε	4	ε	ε	ε
A:	ε	ε	6	9	ε	ε
DH_1:	ε	ε	ε	ε	ε	5
DH_2:	ε	4	ε	ε	ε	ε
R:	ε	8	ε	ε	4	ε

Table 6.1: The communication table of the subject network.

The inbound stations in this table are written in the first column and the outbound stations in the first row. This table can then easily be transformed into the standard adjacency matrix. For the sake of convenience, we include the station corresponding to each row.

$$A_0 = \begin{pmatrix} E: & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 12 \\ U: & 10 & \varepsilon & 4 & \varepsilon & \varepsilon & \varepsilon \\ A: & \varepsilon & \varepsilon & 6 & 9 & \varepsilon & \varepsilon \\ DH_1: & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 5 \\ DH_2: & \varepsilon & 4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ R: & \varepsilon & 8 & \varepsilon & \varepsilon & 4 & \varepsilon \end{pmatrix}$$

Using this standard adjacency matrix, we can construct a periodic regime using the eigenvalues of the above matrix.

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 8 \\ 8 \\ 0 \\ 1 \\ 2 \\ 6 \end{pmatrix} \to \begin{pmatrix} 1 \\ 18 \\ 18 \\ 10 \\ 11 \\ 12 \\ 16 \end{pmatrix} \to \begin{pmatrix} 2 \\ 28 \\ 28 \\ 20 \\ 21 \\ 22 \\ 26 \end{pmatrix} \to \begin{pmatrix} 3 \\ 38 \\ 38 \\ 30 \\ 31 \\ 32 \\ 36 \end{pmatrix} \to \begin{pmatrix} 4 \\ 48 \\ 48 \\ 40 \\ 41 \\ 42 \\ 46 \end{pmatrix} \to \begin{pmatrix} 5 \\ 58 \\ 58 \\ 50 \\ 51 \\ 52 \\ 56 \end{pmatrix} \to \dots$$

The above regime is the timetable of the train network. This means the goal of this chapter is to have the network operate in a manner that resembles the above regime as closely as possible. We see that the initial departure of this timetable is the eigenvector $\mathbf{x}(0) = (8, 8, 0, 1, 2, 6)^T$, which corresponds to the eigenvalue 10, which is a 1-hour-periodic regime.

6.2. Implementing Dynamic Delays

Now that we have established the network and its timetable, we can add delays to the network. We will start by modelling delay onset and delay sustaining. We will then discuss realistic parameters for the probability distributions, concluding by showing some delayed departure sequences.

6.2.1. Modelling Random Delays

The delays present in a network can caused by some new obstacle in the network, but it can also be caused by an obstacle in a previous time step that has not yet been removed. As such, we need to model both random delay onset, but also random delay sustaining. We start with the former. Before we show any of the algorithms, note those shown in this section are simplified versions of the algorithms actually used in the code. The algorithms we discuss here are simply used to clarify the methods used, they are not completely functional.

As discussed in subsection 5.3.3, delays will be added following independent Bernoulli distributions. As such, we define binary random variables for each possible delay:

$$O(TD)(i,j) \sim Ber(p) \qquad O(TD) = \text{Onset Track Delay on arc } (i,j)$$
$$O(SD)(j) \sim Ber(p) \qquad O(SD) = \text{Onset Station Delay in station } j$$
$$O(TF)(i,j) \sim Ber(p) \qquad O(TF) = \text{Onset Track Failure on arc } (i,j)$$

Once these variables are initialised, we can model delay onset. We will also model the sustaining of prior delays into the current time step. To be able to do this, we need to remember which delays were present in the previous time step. To do this, we initialise a dictionary, `TD_dict` and a list `TF_lst`. When there is a track delay onset, the arc will be placed into `TD_dict`, along with the weight of the delay. When there is a track failure, the corresponding arc will be places into `TF_lst`. Since station delays can not sustain, they do not need a list.

Algorithm 1: Delay Onset

```
for outbound in Stations do
    if O(SD)(outbound) then
        A ⟵ station_delayed_matrix(A, outbound, delay_prob)
    for inbound in Stations do
        if O(TD)(inbound, outbound) then
            TD_dict[(inbound, outbound)] = track_delay(inbound, outbound, delay_prob)
        if O(TF)(inbound, outbound) then
            TF_lst.append((inbound, outbound))
```

We note that in this algorithm, it is possible for a train to experience both a track and a station delay. The above algorithm gives us the delayed variant of the adjacency matrix to be applied in this time step, assuming no prior delays sustained to the current time step.

Now there are lists with all present delays that have a chance to sustain. If a new delay arises on an arc that was already delayed, then the new delay is simply the maximum of the two delays to avoid complications. At the beginning of each time step, we again define binary random variables for each possible delay sustaining:

$$S(TD)(i,j) \sim Ber(q) \qquad O(TD) = \text{Sustain Track Delay on arc } (i,j)$$
$$S(TF)(i,j) \sim Ber(q) \qquad O(TF) = \text{Sustain Track Failure on arc } (i,j)$$

Note that the above variables only need to be defined if (i,j) is in `TD_dict` or in `TF_lst` respectively. At the start of the time step, these random variables dictate whether the delay is sustained. If it is not, then the arc (i,j) is removed from the corresponding dictionary or list.

Algorithm 2: Sustaining Delays

```
for (inbound, outbound) in TD_dict do
    if not S(TD)(inbound, outbound) then
        TD_dict.remove((inbound, outbound))
for (inbound, outbound) in TF_lst do
    if not S(TF)(inbound, outbound) then
        TF_lst.remove((inbound, outbound))
```

Now we have a list with all delays present in the current time step, both recent and prior delays. After all other calculations in the time step are done (such as switching to a desirable adjacency matrix), all delays are applied to the current matrix.

Algorithm 3: Applying Delays

```
for delay in TD_dict do
    A ⟵ track_delayed_matrix(A, inbound, outbound, delay)
for delay in TF_lst do
    A ⟵ track_failure_matrix(A, inbound, outbound)
```

Note that algorithm 1 already added the station delay, so this delay does not need to be added again. The order in which the algorithms are to be executed is the following: First, it is determined which delays carry over from the previous time step using algorithm 2, then all other computations of the current time step are done, as said previously. After this, new delays are determined using algorithm 1 and finally, the delays are applied using algorithm 3.

6.2.2. Probabilistic Parameters

In the previous subsection, we initialised numerous binary random variables, as well as some discrete random variables, namely the variables called `delay`. We already discussed that the binary variables were initialised using the Bernoulli distribution. As for the delay variables, we initialise them using the following probability mass function:

$$\mathbb{P}(\texttt{delay} = \delta) = \texttt{delay\_distr}[\delta]$$

Where `delay_distr` is a dictionary containing all possible delays with their probabilities. All the probability parameters are chosen before running the simulation.

We now need to determine which choice of these probability parameters is most suitable for the simulation. One might expect that choosing realistic parameters would be desired, but since we only simulate a single day, this may not be the case. Some days are more eventful than others, and where creating delay resolution method for calm days may be easy, it is the more eventful days where powerful methods are really important. As such, we will choose the probabilistic parameters so that the resulting model has a sufficiently large amount of delays, without them spiralling out of control. This way, the simulation can benchmark how well the method handles busy days with a large amount of delays.

Our network consists of 6 stations and 9 connections. Furthermore, we assume one time unit to be equal to 6 minutes. The equilibrium regime we formulated in section 6.1 repeated every 10 time units, or 1 hour. In order to have an eventful day, we want an average of 2 trains to be delayed every time step. We also want 1 station to be delayed and 1 connection to fail every 2 time steps. We start with the chance of the onset of a station delay, as it can not sustain. We do this by using the formula of the expected value of a Bernoulli distribution.

$$\begin{aligned}\mathbb{E}(\#SD) &= \sum_{j \in \text{stations}} \mathbb{E}(O(SD_i)) \\ &= \#\text{stations} \times p_{O(SD)} \\ &= 6p_{O(SD)}.\end{aligned}$$

We want this number to be equal to $\frac{1}{2}$, so $p_{O(SD)} = \frac{1}{12}$. We now move on to the chance of the onset of a track delay.

$$\mathbb{E}(\#TD) = \sum_{(i,j) \in \text{arcs}} \mathbb{E}(TD_{(i,j)}) \tag{6.1}$$

where the expected value for the presence of a track delay is equal to the chance that a new track delay arises plus the chance that an old track delay carries over. Letting e be the amount of arcs on the communication graph, we get:

$$\begin{aligned}\mathbb{E}(\#TD) &= e \times p_{O(TD)} + e \times p_{O(TD)} \times p_{S(TD)} + e \times p_{O(TD)} \times p_{S(TD)}^2 + \ldots \\ &= e \times p_{O(TD)} \times \sum_{k=0}^{\infty} p_{S(TD)}^k.\end{aligned}$$

Since $p_{S(TD)}$ is a probability, raising it to high powers gives very small numbers, so we neglect each term of the series apart from the first one[1]:

$$\mathbb{E}(\#TD) \approx e \times p_{O(TD)} + e \times p_{O(TD)} \times p_{S(TD)}.$$

We wanted this expected value to be approximately 2. Furthermore, our network has 9 connections, so this equation yields

$$\begin{aligned}2 &= 9 \times p_{O(TD)} + 9 \times p_{O(TD)} \times p_{S(TD)} \\ \frac{2}{9} &= p_{O(TD)}(1 + p_{S(TD)})\end{aligned}$$

[1]The series can be evaluated exactly, but since we only want to estimate probability parameters, this is not necessary.

We choose the values $p_{O(TD)} = \frac{2}{10}$ and $p_{S(TD)} = \frac{1}{9}$ which satisfy the above equation. The same procedure can be applied to the expected amount of track failures, which yields:

$$p_{O(TD)} = \frac{2}{10} \qquad p_{S(TD)} = \frac{1}{9}$$
$$p_{O(SD)} = \frac{1}{12}$$
$$p_{O(TF)} = \frac{1}{20} \qquad p_{S(TF)} = \frac{1}{9}$$

Now that we have established the chances for delays to occur and sustain, we will move on to model the distribution for the probability of delay sizes. For the sake of simplicity, we will use a simplified version of the normal distribution with average delay size 2 and standard deviation 1 for the track delays:

Track Delay Size Distribution	$\lvert\delta_T\rvert \sim N(2,1)$
$\mathbb{P}(\lvert\delta_T\rvert = 1) = 0.25$	$f(\lvert\delta_T\rvert = 1) \approx 0.24$
$\mathbb{P}(\lvert\delta_T\rvert = 2) = 0.5$	$f(\lvert\delta_T\rvert = 2) \approx 0.4$
$\mathbb{P}(\lvert\delta_T\rvert = 3) = 0.25$	$f(\lvert\delta_T\rvert = 3) \approx 0.24$

Table 6.2: The delay size distribution (left) compared to the probability density of the normal $N(2,1)$ distribution (right).

and we will use the following simple distribution for station delays

$$\mathbb{P}(|\delta_T| = 1) = \frac{2}{3} \qquad \mathbb{P}(|\delta_T| = 2) = \frac{1}{3}$$

6.2.3. Example Delayed Sequences

When running the simulation multiple times, the random delays established above will not be the same between runs. This means that any results we derive from a run, may not be reproducible. To solve this issue, we will always be using a seed when running a simulation, to make sure we can reproduce the results. The seed we will use for the random delays is 123, as signified by the line `random.seed(123)` in the python code. Running the simulation with this seed and the random delays enabled yields the delayed departure sequence

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 8 \\ 8 \\ 0 \\ 1 \\ 2 \\ 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 21 \\ 22 \\ 10 \\ 14 \\ 12 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 28 \\ 34 \\ 23 \\ 24 \\ 26 \\ 30 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 42 \\ 38 \\ 36 \\ 35 \\ 38 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 49 \\ 52 \\ 47 \\ 42 \\ 42 \\ 48 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 62 \\ 62 \\ 52 \\ 57 \\ 58 \\ 60 \end{pmatrix} \rightarrow \ldots$$

which corresponds to the delay sequence

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 3 \\ 4 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 0 \\ 6 \\ 3 \\ 3 \\ 4 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 4 \\ 0 \\ 6 \\ 4 \\ 6 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 1 \\ 4 \\ 7 \\ 1 \\ 0 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \\ 4 \\ 2 \\ 6 \\ 6 \\ 4 \end{pmatrix} \rightarrow \ldots$$

As may be apparent, we now run into a problem of notation. Not only are the above departure sequences difficult to interpret, it is also difficult to determine which delays are new and which are a result of pre-existing delays. We will attempt to resolve this issue as follows using 2 measures. The first is that when a train encounters a delay, we write this delay separately from the departure time in the case that the delay had not happened. If a train was scheduled to depart at 5, but was delayed by 2 time units, we write

$$5 + 2.$$

The second measure we take is that whenever a train is still delayed from a previous delay, then we write the departure time in red. If the above train departing at 7 departs from the next station at 9, while he should have departed at 8, we write the departure time

$$9. \quad (6.2)$$

If an old delay causes a late departure and this departure is further delayed by a new delay, then we combine the notation, writing

$$9 + 3.$$

The above notation is not perfect, as the notation in (6.2) does not show the severity of the delay, but it prevents overly messy notation. We now rewrite the delayed departure sequence using the new notation.

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 8 \\ 8 \\ 0 \\ 1 \\ 2 \\ 6 \end{pmatrix} \to \begin{pmatrix} 1 \\ 18+3 \\ 18+4 \\ 10 \\ 11+3 \\ 12 \\ 16 \end{pmatrix} \to \begin{pmatrix} 2 \\ 28 \\ 31+3 \\ 23 \\ 21+3 \\ 26 \\ 30 \end{pmatrix} \to \begin{pmatrix} 3 \\ 42 \\ 38 \\ 39+3 \\ 35 \\ 38 \\ 42-6 \end{pmatrix} \to \begin{pmatrix} 4 \\ 48+1 \\ 52 \\ 44+3 \\ 41+1 \\ 42 \\ 46+2 \end{pmatrix} \to \begin{pmatrix} 5 \\ 60+2 \\ 59+3 \\ 53-1 \\ 53+4 \\ 56+2 \\ 60 \end{pmatrix} \to \dots$$

We observe that in the table above, some delays are negative, for example $\delta_R(3) = -6$.[2] This negative delay is caused by decoupling. A delayed train was decoupled from its inbound station due to a track failure, because of which, the other trains in the station did not need to wait for the delayed train, allowing them to depart sooner.

We see that the delayed departure sequence has not become less intimidating, but it has become easier to spot where new delays arise and how long large combinations of delays last. We can clearly see in the above sequence for example, that the fifth departure is completely delayed; not a single train departed on time. In later sections, we will determine properties of the delayed departure sequence, so that manually studying delayed departure sequences is no longer necessary.

6.3. Applying Delay Resolution

Now that we have added random delays to the model, we can connect the delay resolution methods we devised in chapter 4. We have seen before that the combinatorial and greedy methods are simply specific instances of the p-greedy method and as such, we will apply the p-greedy method with varying values for p. In this section, we will choose $p = 3$, in later sections, we will use various values for p for the sake of benchmarking.

We apply the delay resolution methods to the dynamic simulation as straight-forward as possible. Given a delayed state, the method determines the fastest way to resolve the delay and applies the necessary switches until a new delay occurs.

Method 7 *Given is a multi-switching simulation, a delayed state in this simulation $\tilde{x}(k)$ and a delay resolution method M.*

1. *Set $\tilde{x} = \tilde{x}(k)$*
2. *Determine the resolution sequence $R = M(\tilde{x})$ produced by the delay resolution method.*
3. *Let $R(0)$ be the first entry of the resolution sequence. Switch to matrix $A_{R(0)}$ in the current adjacency class.*
4. *Let the simulation perform one step with $A_{R(0)}$ yielding $\tilde{x}(k+1) = A_{R(0)} + \delta(k)$, where $\delta(k)$ is the delay that occurs in time step k.*

[2]Here we use the notation $\delta_i(k)$ = 'the delay at station i at time unit k'

> 5. *Set $\tilde{x} = \tilde{x}(k+1)$ and return to step 2.*

Since the above method does not have an exit condition, it does not terminate. This is caused by the assumption that train networks do not stop operating. In the event of maintenance or a pause in the network, exit conditions can be added. One might expect the method to terminate once all delays are resolved, but since non-delayed departures can be regarded as delayed departures with delay 0, this is not necessary. One should also notice that while the entire resolution sequence R is calculated in step 2, only its first entry is used, as new delays may change the necessary steps to resolve delays. Through this last remark, we notice that using our concept of score, we can reduce the computation time by only calculating a part of the delay resolution sequence.

> **Method 8** *Given is a multi-switching simulation, a delayed state in this simulation $\tilde{x}(k)$ and a p-greedy delay resolution method M_p.*
>
> 1. *Set $\tilde{x} = \tilde{x}(k)$*
> 2. *Determine the first p terms, $\tilde{R}_1$, of the delay resolution sequence $R = M_p(\tilde{x})$ produced by the delay resolution method.*
> 3. *Let $R(0)$ be the first entry of the resolution sequence segment $\tilde{R}_1$. Switch to matrix $A_{R(0)}$ in the current adjacency class.*
> 4. *Let the simulation perform one step with $A_{R(0)}$ yielding $\tilde{x}(k+1) = A_{R(0)} + \delta(k)$, where $\delta(k)$ is the delay that occurs in time step k.*
> 5. *Set $\tilde{x} = \tilde{x}(k+1)$ and return to step 2.*

This above method only works for iterative delay resolution methods, i.e. methods that produce intermediate results. This means it works for greedy methods, but not the combinatorial method. For such iterative methods, methods 7 and 8 produce the same results.

We now show the results of these delay resolution method. We use the same departure sequence and randomizer seed as in in section 6.2. As for the possible switches, we allow every commute time to be decreased by 1 time unit. If the commute time exceeds 8 time units, a speed up of 2 time units is permitted. This gives the following delayed departure sequence:

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 8 \\ 8 \\ 0 \\ 1 \\ 2 \\ 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 18+3 \\ 18+4 \\ 10 \\ 11+3 \\ 12 \\ 16 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 28 \\ 31+3 \\ 23 \\ 21+3 \\ 26 \\ 29 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 41 \\ 38 \\ 33+3 \\ 34 \\ 38 \\ 41+5 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 48+1 \\ 50 \\ 43+3 \\ 41+1 \\ 42 \\ 46+2 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 60+2 \\ 59+3 \\ 52 \\ 53+4 \\ 54+2 \\ 57 \end{pmatrix} \rightarrow \ldots$$

Though at first glance, no noticeable changes are present, when looking at the total delay per time step, we do notice a significant difference.

k	0	1	2	3	4	5	6	7	8	9	10	Cumulative
No Resolution	0	10	20	20	15	26	33	33	36	36	38	267
3-composite	0	10	19	18	12	21	26	24	23	24	23	200

Table 6.3: The total delay per time step.

We see here that at time step 10, the network using the resolution method has 15 time units, or 90 minutes of delay less than when no resolutions are attempted. Furthermore, over these 10 time steps, the 'No Resolution' network has accumulated 67 time units, or more than 6 hours of delays, more than its '3-composite' counterpart. We do see that some severe delays still persist. This becomes abundantly apparent when looking at the delay sequence of the above delayed departure sequence.

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 3 \\ 4 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 0 \\ 6 \\ 3 \\ 3 \\ 4 \\ 3 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 3 \\ 0 \\ 6 \\ 3 \\ 6 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 1 \\ 2 \\ 6 \\ 1 \\ 0 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 4 \\ 4 \\ 2 \\ 6 \\ 4 \\ 1 \end{pmatrix} \rightarrow \dots$$

Here we see that at time step $k = 5$, some departures are delayed up to 6 time units, or 36 minutes. In order to prevent delays of this scale, we set the following decoupling condition:

If a train arrives at its inbound station ***equal or more than half its commute time late****, then other trains at the station do not need to wait for it.*

For example, if a train is supposed to arrive at 16 at a station after an 8 time unit commute, but the train arrives at 20 due to delays, other trains do not have to wait for it. When adding this to the simulation, we get the following delay sequence:

$$\begin{pmatrix} k \\ E \\ U \\ A \\ DH_1 \\ DH_2 \\ R \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 3 \\ 4 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 0 \\ 0 \\ 3 \\ 3 \\ 4 \\ 3 \end{pmatrix} \rightarrow \begin{pmatrix} 3 \\ 2 \\ 0 \\ 6 \\ 3 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 4 \\ 1 \\ 0 \\ 6 \\ 1 \\ 0 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 5 \\ 3 \\ 0 \\ 2 \\ 6 \\ 2 \\ 0 \end{pmatrix} \rightarrow \dots$$

With this, we can once again create a table showing the total delay per time step.

k	0	1	2	3	4	5	6	7	8	9	10	Cumulative
No Resolution	0	10	20	20	15	26	33	33	36	36	38	267
3-composite	0	10	19	18	12	21	26	24	23	24	23	200
3-C, decoupling	0	10	13	11	11	19	19	16	16	19	10	144

Table 6.4: The total delay per time step.

We see here that the cumulative delay has further decreased and in almost every time step, the total delay was reduced. From the results, this method seems to be much better than the other two, but one needs to keep in mind that decoupling can be a significant inconvenience to many travellers, so this method may not always be viable.

6.4. Speed Up and Decoupling Restrictions

During some parts of the day, speed ups and decoupling may not be possible in a network due to obstacles that present at a certain time. In order to properly translate this in the simulation, we first need a notion of time. We already assumed that each time unit corresponds to 6 minutes, and now we will also introduce a start and end time for the train network. We set these two times to be 6 in the morning and 10 in the evening, allowing for 16 operational hours. Since the timetable is 10-periodic, we assume that each time step corresponds with 1 hour. We do not account for any delays in this assumption in order to avoid further complexity. Now in order to realistically simulate a day's worth of trains, we implement the following speed up and decoupling restrictions:

- From 6:00 to 10:00 and from 20:00 to 22:00 (time steps 0-4 and 14-16) only speed ups of 1 time unit are allowed.

- During rush hour, from 8:00 to 10:00 and from 17:00 to 19:00 (time steps 2-4 and 11-13), no decoupling is allowed
- Once per day, at 13:00 (time step 7), all networks are occupied by freight trains, so no speed ups are allowed.

Using the 3-composite greedy method with decoupling, we get the following results when adding the above restrictions:

k	0	1	2	3	4	5	6	7	8	...	16	Cumulative	Decoupling
No Restrictions	0	10	13	11	11	19	19	16	16	...	8	200	8
Restricted	0	10	13	11	11	19	19	17	17	...	14	210	8

Table 6.5: The total delay per time step using the 5-composite greedy method with decoupling conditions.

We see that the extra restrictions do not have a major impact on the results, though we should keep in mind that the restrictions cause an additional full hour of delays over the entire day. Such delays are not to be underestimated in our real-world context.

6.5. Resolution Evaluation Criteria

We now have all building blocks required to start evaluating delay resolution methods. We simulated a full day of train traffic, connected the resolution methods to the simulation and added resolution restrictions to make the simulation more realistic. In previous sections, we have already used a few properties of simulation runs to study changes in the simulations. These were the total delays per time unit, the cumulative delay over the entire day and the amount of decoupling in a day. In this section, we will introduce some more of these simulation properties, which we will henceforth call simulation statistics.

We start by defining some simple simulation statistics based on the delay sequence.

- **Maximum Delay per time unit:** the largest delay of a departure in a single time unit k,[3]

$$\max_{i \in \mathcal{M}} \delta_i(k)$$

- **Maximum delay per station:** the largest delay at a station i from $k = 0$ up to and including some time step $k = n$,

$$\max_{0 \le k \le n} \delta_i(k)$$

- **Cumulative Delay per Station:** the sum of all delays at a station i from $k = 0$ up to and concluding some time step $k = n$,

$$\sum_{k=0}^{n} \delta_i(k)$$

In the tables in the previous chapters, we repeatedly used the 'total delay' per time unit. Condensing an entry in the delay sequence into a single number makes it easier to study the behaviour of the delay, so we will continue to consider this statistic. For the sake of simplicity, we will henceforth denote the total delay per time unit as $\Delta(k) = \sum_{i \in \mathcal{M}} \delta_i(k)$. Using this notation, we can now define some simulation statistics.

- **Cumulative Total Delay (CTD):** The sum of all total delays from $k = 0$ up to and concluding some time step $k = n$,

$$\sum_{k=0}^{n} \Delta(k)$$

[3]We use the notation that $i \in \mathcal{M}$ if i is a station in the network corresponding to $\mathcal{M}$.

- **Maximum Total Delay (MTD):** The maximum total delay from $k = 0$ up to and including some time step $k = n$,

$$\max_{0 \le k \le n} \Delta(k)$$

- **Mean Total Delay:** The maximum total delay divided by the amount of time steps. The first time step $k = 0$ is not taken into account as its total delay is always 0.

$$\bar{\Delta} = \frac{1}{n} \sum_{k=1}^{n} \Delta(k)$$

- **Standard Deviation of the Total Delay (SD):** this can be used to measure how consistent the total delay is across time steps.

$$SD(\Delta) = \sqrt{\frac{1}{n-1} \sum_{k=1}^{n} (\Delta(k) - \bar{\Delta})^2}$$

The above statistics can be used to perform in-depth analyses of delay resolution methods. These analyses can then be used to fine-tune resolution methods. Performing such analyses and fine-tuning are outside the scope of this report, and as such, we will perform some simpler analyses using only the latter 4 statistics for the sake of comparing the delay resolution methods we formulated.

6.6. Results

We now set out to evaluate the p-composite greedy method for various values of p, to determine the optimal value. We run the simulation for the values $p = 0, 1, 2, 3, 4, 5$, where $p = 0$ corresponds to not intervening in the delay propagation. The results can be found in table 6.6.

p	CTD(p)	MTD(p)	SD(Δ)(p)	Decoupled	Time(p)
0	222	26	6.34	9	0.015s
1	244	24	6.26	6	0.018s
2	186	21	5.08	6	0.069s
3	210	19	4.97	8	0.979s
4	232	23	6.10	7	13.174s
5	212	19	4.98	8	146.614s

Table 6.6: Various statistics for the p-composite greedy delay resolution method with varying values for p. The used seed is 123.

Since the simulation has a heavy random element, results based on a single run are of little statistical value. As such, we conduct 100 such runs for each value of p. The seeds we use for these runs are $0, 1, 2, \ldots, 99$. We then take the average of each statistic over the different runs to yield results with a higher statistic relevance.

p	CTD(p)	MTD(p)	SD(Δ)(p)	Decoupled
0	165.89	22.12	6.22	4.81
1	144.33	18.82	5.34	3.04
2	139.32	17.97	5.10	2.55
3	138.06	17.56	5.07	2.37
4	134.11	17.18	4.96	2.58

Table 6.7: Average statistics of 100 runs of the simulation with various values for p.

In the above table, we do not show the computation time, as table 6.6 clearly shows that the computation time quickly ramps up, which is consistent with our expectations. Any results beyond that observation are of little substance. We also omitted $p = 5$ from the above table due to long computation times. Of interest is that tables 6.6 and 6.7 tell very different stories. In table 6.7, we see that as p increases, the total cumulative delay decreases. This result is not entirely unexpected as higher value of p essentially correspond to a 'smarter' delay resolution method. In table 6.6, a very different result can be observed. In this table, $p = 2$ is superior in cumulative total delay and decoupled trains compared to the other values of p. This implies that being able to look further into the future is not always beneficial to a greedy method.

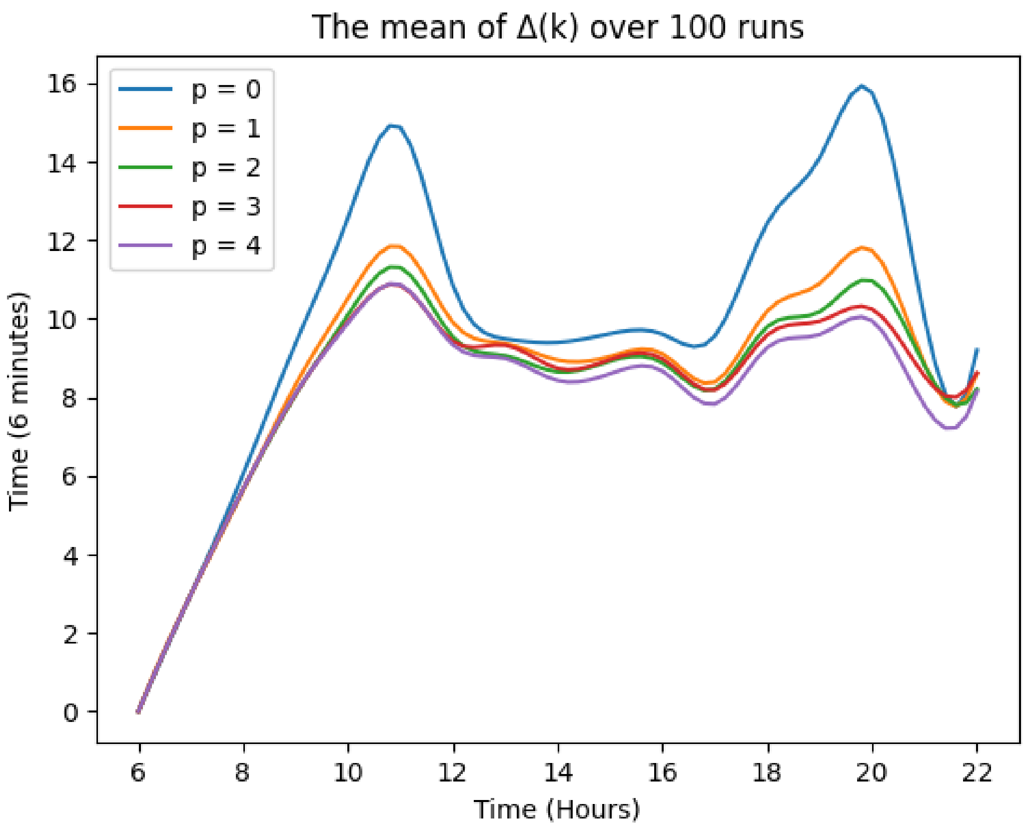


Figure 6.3: The average total delay per hour over 100 samples. The x-axis represents the time of day and the y-axis the average total delay. Higher values of p are beneficial, but this difference decreases as p gets larger.

We can conclude that in general, higher values for p, and thus looking further into the future, benefits the p-greedy delay resolution method. This benefit is more significant for smaller values for p and diminishes for very large values of p. The computation times for the method become increasingly bigger for larger values of p and at $p = 5$, we deem the computation time sufficiently large to stop increasing p further. From the single run corresponding to table 6.6, we can observe that the general result does not necessarily hold for specific cases. It is thus possible that in specific circumstances, specific smaller values of p yield better results than large values.

The last assertion is an interesting topic for further study. Using the simulation, one might be able to pinpoint specific criteria for which a value of p is more optimal. In addition to this, changing the probability parameters may have a significant effect on the results of the simulation. As it is not within the scope of this report, we will not dabble in such analyses, but the reader is warmly invited to seek such results for themselves.

7 Conclusion

In this final chapter, we sum up the results written down in this report. The obtained results are split into 3 parts: max-plus modelling, delay resolution and network simulation. Each of these parts built on the foundations of the previous, but also came with results of their own, which we will discuss in detail. We will conclude this chapter with a brief summary of these results and a list of possible further topics of research based on the achieved results.

7.1. Max-Plus Modelling Results

In chapter 2, we gave the basics of max-plus algebra and max-plus modelling. This chapter acted as a necessary mathematical foundation to build upon. The mathematics in this chapter are heavily based on the book Max Plus at Work (Heidergott et al., 2006). In chapter 3, we built upon the basic max plus model in the form of switching. The concept of switching was based on the bachelor thesis 'Control of Delay Propagation in Railway Networks Using Max-Plus Algebra' (Hoekstra, 2020). The aim of the chapter was to introduce a more formal notation for switching systems in order to make them more suitable for more sophisticated mathematical procedures and extensions. In chapter 5, we expanded the switching model to account for changing network environments, such as network obstructions or switching restrictions. Adding this extension made the model more realistic and thus more widely applicable.

The achieved end result of the modelling component of this thesis is the ability to convert logistic networks, such as train networks, into mathematical models. These models can account for controlled changes such as speed ups through switching and they can account for uncontrolled changes such as network obstructions through shifting as introduced in the multi-switching model.

7.2. Delay Resolution Results

Delays were introduced in chapter 3, with the formal formulation of the delay problem in chapter 4. In this latter chapter, we immediately formulated 3 methods for resolving delays: the Combinatorial Method, the Greedy Delay Resolution Method and the p-Composite Greedy Delay Resolution Method. The first two methods are special cases of the p-greedy method for certain values of p. We found that for smaller values of p, the p-greedy method is faster, but worse at resolving delays, in some cases even unable to resolve delays that can in fact be resolved. For larger values of p, the method becomes better at resolving delays but significantly slower. We also explored the possibility of decoupling trains and formulated possible conditions for decoupling. We found that decoupling can have a massively beneficial effect on delay resolution, though it may prove an inconvenience for commuters in the network. In chapter 5, we also introduced the concept of scores. These scores offered a way to evaluate the quality of a departure (or state) and helped the resolution methods resolve the delays.

7.3. Network Simulation Results

In chapter 6, we were able to create a simulation for a train network containing some of the major dutch cities. We were able to implement the multi-switching model and delay resolution methods along with delay resolution restrictions. We implemented dynamic random delay to make the simulation realistic. In the simulation, we saw that applying the p-composite greedy delay resolution method reduces the delay and in general, higher values for p yield better results. We also saw however, that this is not always the case and in specific cases, specific, lower values for p yield better results.

7.4. Overall Results

If there is one thing the reader should take away from this report, it is the following. The existing max-plus model has been extended to account for speed ups and network restrictions. Train delays have been studied and delay resolution methods have been designed. Finally, a simulation was made to study the behaviour of train networks while delays are added at random. This simulation has also been used to evaluate the delay resolution methods.

7.5. Further Research

Like the concluding results, we will discuss the potential further research for each aspect of the report separately. In chapter 4, we already mentioned the importance and complexity of decoupling. Though appendix B was dedicated to discussing this topic, many of its implications are still left untouched. A potential topic for further research is the modelling of max-plus systems where trains can be decoupled and recoupled without altering the network or disappearing from it entirely. Such a modelling method would allow for improved modelling of rush hours or other modified timetables. This would allow logistic networks to operate more dynamically, changing the timetables when the situation calls for it.

Concerning delay resolution, there are two possible directions for research. The first is studying delay resolution methods. One could look into the behaviour of the methods discussed in this report to determine strong and weak points to improve the methods. One could also think of other clever ways to resolve delays and make a new resolution method based on these ideas. The other direction is studying the score problem introduced in chapter 5. One could choose different scores to determine if there is a better score criterion to resolve delays. This process of choosing scores could also be done using machine learning algorithms through reinforcement learning.

For the simulation, there are 2 potential topics for further research as well. The first is to use the current simulation to determine properties and behaviour of delays and the delay resolution methods. As said before, in some situations, specific, lower values of p perform better than higher values. Determining which underlying process determines the optimum may provide some further insights into the workings of dynamically delayed max-plus systems. The other potential direction for research is to extend the current simulation to include rush hour or even external processes, such as flight transfers or external trains. Especially the latter may prove to be interesting topics as modelling interactions between the subject network and other, external networks, would allow larger networks to be modelled separately and then be connected using this interaction modelling.

In appendix C, we linked the max-plus models discussed in this report to conventional systems and control theory, mentioning some topics such as state feedback, predictive controllers and repetitive controllers. Researching to which extend max-plus models and systems and control theory are connected may be another interesting topic for study. Performing this research may also bring to light how existing methods from conventional systems theory could help to solve problems such as the delay and scoring problems.

Bibliography

Breckon, D. (n.d.). Cover image: Great western 5037 leaving reading station. https://www.courtenaysfineart.com/Don_Breckon.html

Facts and figures 2016. (2016). https://www.deutschebahn.com/en/facts_figures-6929164#:~:text=In%5C%20the%5C%20Passenger%5C%20transport%5C%20division,nearly%5C%2012%5C%20million%5C%20passengers%5C%20daily.

Heidergott, B., der, W. J. v., & Olsder, G. J. (2006). *Max plus at work: Modeling and analysis of synchronized systems*. Princeton University Press.

Hoekstra, M. (2020). Control of delay propagation in railway networks using max-plus algebra. http://resolver.tudelft.nl/uuid:b5816386-0ed9-4760-a6de-f0db0c3a5226

Hong, J., Chu, Z., & Wang, Q. (2011). Transport infrastructure and regional economic growth: Evidence from china. *Transportation*, *38*(5), 737–752.

Middelkoop, A. (2022). Zondag 3 april: Tot 20.00 uur geen treinverkeer. https://nieuws.ns.nl/zondag-3-april-vanmiddag-geen-ns-treinen/

Ramos, G. A., Costa-Castelló, R., & Olm, J. M. (1970). Repetitive control. https://link.springer.com/chapter/10.1007/978-3-642-37778-5_2

Schwenzer, M., Ay, M., Bergs, T., & Abel, D. (2021). Review on model predictive control: An engineering perspective - the international journal of advanced manufacturing technology. https://link.springer.com/article/10.1007/s00170-021-07682-3#citeas

A

Residual Proofs and Derivations

In this appendix, we will show some derivations that were omitted from the main report due to being to long or not immediately relevant. We will also proof all theorems given in this report, as well as give some additional intuitions.

A.1. Max-Plus Algebra

In chapter 2, an important theorem was given that says that a departure sequence of a max-plus model is entirely characterised by the base of its initial departure (and the adjacency matrix)

Theorem 1 *Let* $(\mathbf{x}(k))_{k\geq 0}$ *be the departure sequence of* $\mathcal{M}(A, \mathbf{x}(0))$.
The commute sequence and base sequence are not changed by translation: $\forall c \in \mathbb{R}$:

$$(d(\mathbf{x}(k)))_{k\geq 0} = (d(\mathbf{x}(k) \otimes c))_{k\geq 0}$$
$$(\lfloor \mathbf{x}(k) \rfloor)_{k\geq 0} = (\lfloor \mathbf{x}(k) \otimes c \rfloor)_{k\geq 0}$$

This means the behaviour of a departure sequence is characterised by the base of its initial departure.

Proof. Let $(\mathbf{x}(k))_{k\geq 0}$ be a departure sequence. Consider the translation of this sequence $(\mathbf{y})_{k\geq 0} = (\mathbf{x}(k) \otimes c)_{k\geq 0}$ for some $c \in \mathbb{R}$. Consider the commute sequence of this latter departure sequence:

$$\begin{aligned} d(\mathbf{y}(k)) &= \mathbf{y}(k+1) - \mathbf{y}(k) \\ &= (\mathbf{x}(k+1) \otimes c) - (\mathbf{x}(k) \otimes c) \\ &= (\mathbf{x}(k+1) + c) - (\mathbf{x}(k) + c) \\ &= \mathbf{x}(k+1) - \mathbf{x}(k) \\ &= d(\mathbf{x}(k)) \end{aligned}$$

Thus proving the first assertion. As for the second assertion, we know that

$$\lfloor \mathbf{x}(k) \rfloor = \mathbf{x}(k) - ||\mathbf{x}(k)||_{min}$$

But we know that $||\mathbf{x}(k) \otimes c||_{min} = ||\mathbf{x}(k)||_{min} + c$, therefore

$$\begin{aligned} \lfloor \mathbf{y}(k) \rfloor &= \mathbf{y}(k) - ||\mathbf{y}(k)||_{min} \\ &= (\mathbf{x}(k) + c) - (||\mathbf{x}(k)||_{min} + c) \\ &= \mathbf{x}(k) - ||\mathbf{x}(k)||_{min} \\ &= \lfloor \mathbf{x}(k) \rfloor \end{aligned}$$

Thus proving the second assertion. □

The reason why this theorem implies that the behaviour of a departure sequence is characterised by the base of its initial departure is that if we translate a departure sequence such that its initial departure is a base, both the commute sequence and the base sequence remain unchanged. These two sequences exactly show the behaviour of a sequence.

We now consider a theorem that formulates an important properties of the max-plus model.

Theorem 2 *Any departure sequence of a max-plus model is causal and forgetful. In other words, the next entry in a sequence is dependent on the current entry, but not on any prior or future entries.*

Proof. The proof of this theorem follows from the fact that a max-plus model is defined with a recurrence relation. In this recurrence relation, we have that $\mathbf{x}(k+1) = A \otimes \mathbf{x}(k)$, where it is clear to see that no past or future terms appear. □

The above theorem is not difficult to proof, but the theorem is extremely important as it also holds for switching systems. This means that when making choices in a switching system, we do not need to consider past or future entries (we can if this is desired, but this much complicates the system).

A.2. Switching Max-Plus

In this chapter, only one theorem was proven. This theorem shows a result for switching max-plus system where the standard index sequence is not constant.

Theorem 3 *If the index sequence J of a switching max-plus model M_S on a strongly connected graph is m-periodic, then the eigenvector $\mathbf{v}$ of*

$$A = \bigotimes_{i=1}^{m} A_{J(m-i)}$$

induces an m-periodic regime with average commute time $\frac{\lambda}{m}$, where λ is the eigenvalue of A associated with $\mathbf{v}$.

Proof. Let $\mathcal{A}$ be an adjacency matrix and J an index sequence drawing matrices from $\mathcal{A}$. Let $j_1, \ldots, j_m$ be the repeating elements in J. Let $\lambda, \mathbf{v}$ be the eigenvalue-eigenvector pair of

$$A = \bigotimes_{i=1}^{m} A_{J(m-i)}$$

This means that A and $\mathbf{v}$ induce a departure sequence where each time step causes a translation of the departure with magnitude λ. So the induced sequence with initial departure $\mathbf{v}(0) = \mathbf{v}$ is

$$\begin{aligned}(\mathbf{v}(k))_{k\geq 0} &= \mathbf{v}(0) \to \mathbf{v}(1) \to \mathbf{v}(2) \to \ldots \\ &= \mathbf{v} \to \mathbf{v} + \lambda \to \mathbf{v} + 2\lambda \to \ldots\end{aligned}$$

Now let $\mathbf{x}_0$ be the initial departure of a departure sequence induced by J with $\mathcal{A}$. This means that

$$\begin{aligned}\mathbf{x}(k+m) &= A_{j_m} \otimes \mathbf{x}(k+m-1) \\ &= A_{j_m} \otimes A_{j_{m-1}} \otimes \mathbf{x}(k+m-2) \\ &= \ldots \\ &= \bigotimes_{i=1}^{m} A_{J(m-i)} \otimes \mathbf{x}(k)\end{aligned}$$

This means that if we let $\mathbf{x}_0 = \mathbf{v}$, then

$$\begin{aligned}\mathbf{x}(0) &= \mathbf{v}\\ \mathbf{x}(m) &= \mathbf{v}(1) = \mathbf{v} + \lambda\\ \mathbf{x}(2m) &= \mathbf{v}(2) = \mathbf{v} + 2\lambda\\ &\vdots\\ \mathbf{x}(km) &= \mathbf{v}(k) = \mathbf{v} + k\lambda\end{aligned}$$

So $\mathbf{x}(km) = \mathbf{x}((k-1)m) \otimes \lambda$, which implies that the departure sequence induced by $\mathbf{x}(0) = \mathbf{v}$ and J with $\mathcal{A}$ is an m-periodic regime with average commute $\frac{\lambda}{m}$. □

A.3. Delay Problems

The first theorem in this chapter revolved around convergence of delay sequences.

Theorem 4 *Let $\mathcal{A}$ be an adjacency class, $(\mathbf{x}(k))_{k\geq 0}$ and $(\mathbf{y}(k))_{k\geq 0}$ departure sequences induced by J, a choice function generating an index sequence and some initial departures $\mathbf{x}(0)$ and $\mathbf{y}(0)$ respectively. Suppose that $J = J(\mathbf{x})$, so the choice function only depends on the current state, then the following hold:*

- *If there is a time step n where the departure times $\mathbf{x}(n) = \mathbf{y}(n)$, then the two departure sequences are equal in every time step after n.*

  $$\text{If } \exists n \in \mathbb{N} : \mathbf{x}(n) = \mathbf{y}(n) \text{ then } \forall k \geq n : \mathbf{x}(k) = \mathbf{y}(k)$$

- *Suppose $(\mathbf{x}(k))_{k\geq 0}$ and $(\mathbf{y}(k))_{k\geq 0}$ enter s-periodic regimes with onset n and m respectively. Let $k \geq \max(n, m)$, if $\mathbf{x}(k) \neq \mathbf{y}(k)$, then the two departure sequences at no point coincide.*

Proof. The first assertion follows from the following observation:

$$\begin{aligned}&\mathbf{x}(n) = \mathbf{y}(n)\\ \implies\ &J(\mathbf{x}(n)) = J(\mathbf{y}(n))\end{aligned}$$

Letting $J(\mathbf{x}(n)) = j$, it holds

$$\mathbf{x}(n+1) = A_j \otimes \mathbf{x}(n) = A_j \otimes \mathbf{y}(n) = \mathbf{y}(n+1)$$

Which inductively holds for every future sequence entry. From this we can conclude that both sequence starting from the index m and n respectively must be the same.

We now move on to the second assertion. Let $(\mathbf{x}(k))_{k\geq 0}$ and $(\mathbf{y}(k))_{k\geq 0}$ enter into s-periodic regimes with onset n and m respectively. Let $M = \max(m, n)$, $k \geq M$ and $\mathbf{x}(k) \neq \mathbf{y}(k)$. Suppose by contradiction that the two sequences converge to one another. Consider only the time steps $t \geq M$. This means that.

$$\begin{aligned}\mathbf{x}(M) &= \mathbf{x}(M + ls) & \mathbf{y}(M) &= \mathbf{y}(M + ls)\\ \mathbf{x}(M+1) &= \mathbf{x}(M + 1 + ls) & \mathbf{y}(M+1) &= \mathbf{y}(M + 1 + ls)\\ \mathbf{x}(M+2) &= \mathbf{x}(M + 2 + ls) & \mathbf{y}(M+2) &= \mathbf{y}(M + 2 + ls)\\ &\vdots\ \vdots & &\vdots\ \vdots\\ \mathbf{x}(M+s-1) &= \mathbf{x}(M + s - 1 + ls) & \mathbf{y}(M+s-1) &= \mathbf{y}(M + s - 1 + ls)\end{aligned}$$

If the two sequences converge, there is a time step $T \geq 0$ so that $\forall t \geq T : \mathbf{x}(t) = \mathbf{y}(t)$. Let $q = s - T(\mod M)$, then $\mathbf{x}(k+T+q) = \mathbf{y}(k+T+q)$ and $T+q$ is a multiple of s, so $T+q = ls$ for some l. This means that $\mathbf{x}(k) = \mathbf{x}(k + ls) = \mathbf{y}(k + ls) = \mathbf{y}(k)$, which contradicts the assumption that $\mathbf{x}(k) \neq \mathbf{y}(k)$. □

B

Decoupling

In chapter 4, we briefly discussed decoupling without going into detail. In this appendix, we will combine the intuition discussed there with the new addition of multi-switching to consider the implications of decoupling. We will reintroduce decoupling using an example and refresh some important definitions. We will then discuss when decoupling should be used to resolve delays. Decoupling can also be used to remove or add trains in the network, which we will illustrate by modelling rush-hour. We will conclude this appendix by discussing some modelling restrictions of decoupling.

B.1. Severe Delays

As seen in the identically named subsection 4.3.1, the need for decoupling arises in the case of large delays. We will thus give an example of such an instance.

Example 25 *Consider the multi-switching max-plus model* $\mathcal{M}_M = \mathcal{M}_M(\Omega, (1,0)^T, \mathcal{H}, J)$ *corresponding to the following network:*

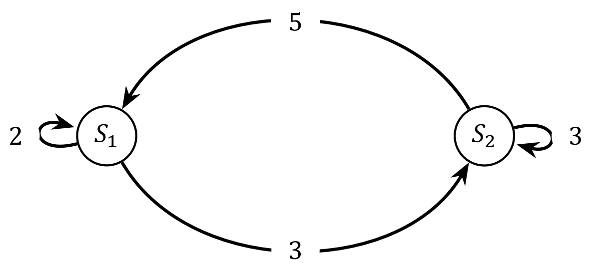


Figure B.1: The communication graph of the example train network

Where the standard adjacency matrix corresponding to the weights in the graph are repeatedly applied ($J = 0,0,0,\ldots, \mathcal{H} = 0,0,0,\ldots$). *This gives the departure sequence*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 5 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 8 \end{pmatrix} \to \begin{pmatrix} 13 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 16 \end{pmatrix} \to \begin{pmatrix} 21 \\ 20 \end{pmatrix} \to \ldots$$

Due to a delay, the train travelling from S_2 *to* S_1 *takes 16 time units instead of 5. If nothing is done to amend this delay, the resulting delayed departure sequence is*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 16 \\ 4 \end{pmatrix} \to \begin{pmatrix} 18 \\ 19 \end{pmatrix} \to \begin{pmatrix} 24 \\ 22 \end{pmatrix} \to \begin{pmatrix} 27 \\ 27 \end{pmatrix} \to \begin{pmatrix} 32 \\ 30 \end{pmatrix} \to \ldots$$

Since the train in the middle upper arc is the only train that is severely delayed, we can simply allow the other train in station S_1 *to depart without waiting for the delayed train. This decoupling yields*

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \begin{pmatrix} 3 \\ 4 \end{pmatrix} \to \begin{pmatrix} 9 \\ 7 \end{pmatrix} \to \begin{pmatrix} 12 \\ 12 \end{pmatrix} \to \begin{pmatrix} 17 \\ 15 \end{pmatrix} \to \begin{pmatrix} 20 \\ 20 \end{pmatrix} \to \ldots$$

Since this delayed departure sequence is ahead of the expected departure sequence, we can simply slow down the necessary trains such that it converges.

We can see that for severe simple delays, decoupling is an easy resolution method. As we will see later in this appendix however, it comes with some aspects that need to be dealt with carefully. Before continuing, we will briefly remind ourselves of some definitions surrounding decoupling.

Definition 24 (15) *Let $\mathcal{M}_S = \mathcal{M}_S(\mathcal{A}, \mathbf{x}_0, J)$ be a max-plus switching model. If the communication graphs $G(A_i)$ of the adjacency matrices in $\mathcal{A}$ do not all have the same arcs, then $\mathcal{M}_S$ is called a decoupled max-plus model. Let $\mathcal{A}_0$ be the set of all matrices that have the same arcs as A_0, we call this set the standard adjacency class and $\mathcal{A}_\varepsilon = \mathcal{A} \setminus \mathcal{A}_0$ the decoupled adjacency class.*

Definition 25 (16) *We define $\mathcal{E}^{(i,j)}$ to be the matrix with all entries equal to 0 except for entry (i, j) which is equal to ε. We call this matrix the (i, j)-decoupling matrix.*

We call $A + \mathcal{E}^{(i,j)}$ the (i, j)-decoupled matrix of A. In the above example, we used the (S_1, S_2)-decoupled matrix to decouple the delayed train. In subsection 5.3.4, we showed that instead of implementing every possible delayed adjacency matrix, we could define a function that would map adjacency matrices to their delayed form. We can use this same method to prevent having to implement every possible decoupling, by using a function that maps the network to a specific decoupled variant of it. We define this decoupling function as follows:

$$\mathcal{E}^{(i,j)}(A) = A + \mathcal{E}^{(i,j)}$$

This function can be repeatedly applied to decouple several trains.

We have also seen that decoupling can happen as a result of for example track failure. Since we have no control over this type of decoupling, it will not be a major topic in this appendix. If such a delay occurs over a long period of time, alternative transport options such as buses may need to be provided, but we will not consider the implications of such changes.

B.2. Decoupling Criteria

Since decoupling is detrimental to the logistic functionality of networks as they disconnect networks, it is important to only allow for decoupling if this is strictly necessary or sufficiently beneficial. In section 4.3, we introduced 3 conditions for decoupling. In this section, we had not established the concept of score. This means we can now reformulate these criteria in a more mathematical manner.

- '*If a delay can not be resolved without decoupling, then the use of decoupling is allowed.*'
- '*If the delay resolution time is too great, then we can allow decoupling to reduce this time.*'

$$\text{Decouple if } RT(\delta) \geq C$$

- '*if decoupling strongly reduces the resolution time, then it is also allowed.*'

$$\text{Decouple if } RT(\delta) \geq RT_D(\delta) + C$$

In the above conditions C is a constant δ is a delay and $RT(\delta)$ is the resolution time of the delay. The subscript D corresponds to decoupling.

- '*If the delay is too great, then we can allow decoupling.*'

$$\text{Decouple if } s(\tilde{\mathbf{x}}) \geq C$$

- '*if allowing for decoupling in one iteration of the method results in a greatly reduces score, then allow decoupling.*'

$$\text{Decouple if } s(M \otimes \tilde{\mathbf{x}}) \geq s(M_D \otimes \tilde{\mathbf{x}}) + C$$

In the above conditions, we determine the size of the delay by the score $s(\tilde{x})$. $M \otimes \tilde{x}$ is the result of one iteration of the given method starting from $\tilde{x}$.

Other decoupling criteria may be desirable, but this is another instance where the used criteria should be determined on a case by case basis.

B.3. Rush Hour Modelling

In some train networks, the amount of trains is not always constant. An example of this is the occurrence of rush hour, during which, the amount of (train) traffic can be increased in order to accommodate for more people commuting. In a lot of networks, rush hour occurs twice per day, once in the morning when people travel to work and once in the evening when people travel back home. We will be introducing several ways to model rush hour, discus transitioning into rush hour and discuss some required model changes.

B.3.1. Modelling Rush Hour

In order to illustrate the various ways to model rush hour in a train network, we will use the following example network:

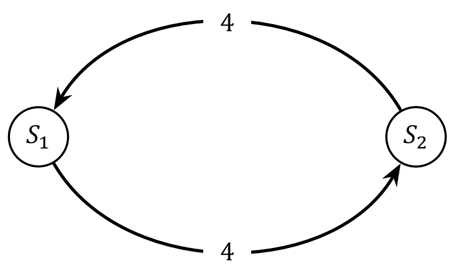
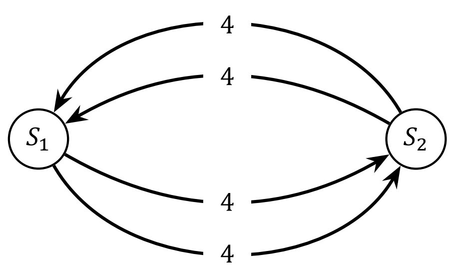


Figure B.2: The communication graph of the example train network

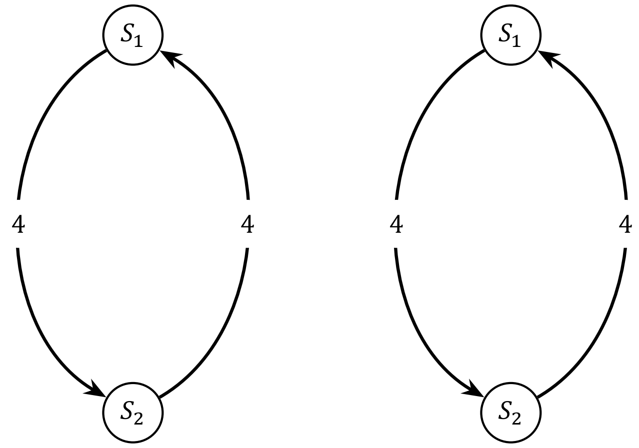
It is clear to see that a periodic regime for this network would be

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 4 \\ 4 \end{pmatrix} \to \begin{pmatrix} 8 \\ 8 \end{pmatrix} \to \begin{pmatrix} 12 \\ 12 \end{pmatrix} \to \begin{pmatrix} 16 \\ 16 \end{pmatrix} \to \begin{pmatrix} 20 \\ 20 \end{pmatrix} \to \begin{pmatrix} 24 \\ 24 \end{pmatrix} \to \dots$$

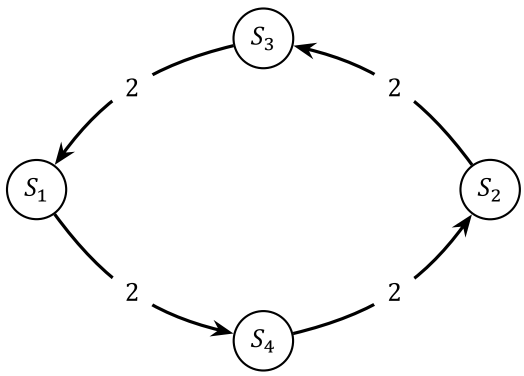
Which means every 4 time units a train leaves both stations. Now suppose that during rush hour, it is desirable that a train leaves every 2 time units. One way to achieve this, is to simply add more trains to the network. Since the model accounts for one train per arc however, this means we would have to add more arcs. We can do this by simply doubling up certain arcs:



Since we want the trains to depart in a staggered pattern, we do not want all of them to wait for each other, meaning we need to decouple some trains. The resulting communication graph is isomorphic to the following graph:



So we see the graph is no longer strongly connected. Another method to add more arcs is to also add more nodes. Adding an extra station on each arc would also add an extra arc and thus an extra train:



But now we have the problem that when rush hour is over, the added train remains. We can resolve this issue by adding extra arcs between S_1 and S_2 with weight 4, but this would needlessly complicated the network. Instead, we can just let the excess trains traverse the theoretical network while removing the physical trains from the tracks. This results in 'ghost trains', trains that exist in the simulation, but not in reality. These ghost trains are not a problem, as they can not interact with other trains. As such, this latter solution is sound method for modelling rush hour.

B.3.2. Transitioning to Rush Hour

When transitioning into and out of rush hour, we are adding trains to the train tracks. It may however not be possible for a train to immediately be at the location where it is desired. In order to make sure that the network is fully operational during rush hour, we need to ensure that all trains are at their desired location. The simplest way to do this is to have room for storing a train at each station. This is however not always possible and quite expensive. Instead, it would be easier to have central points in the network where trains are stored. Right before rush hour, a transition period can then be used to get all trains to their desired positions.

Since the simulation already accounts for the rush hour trains at any time, we can simply allow these trains to match the simulation. This is essentially turning ghost trains into real trains in the network. If we do this, we do need to make sure that the trains start to follow their rush hour regime early enough, to make sure they are not too late for rush hour commuters.

B.3.3. Method Parameters

It is important to note that rush hour is a very busy period on the train network in terms of both trains and passengers. As such, it may not suffice to use the same criteria for the delay resolution methods as normally. It may for example be wise to change the criteria for decoupling, as decoupling would inconvenience more passengers. These kind of changes can be seen as parametric changes; we change the choices made by the methods without changing the underlying functionality. Which changes to make should be decided on a case by case basis, but some options are to change the score function or change the thresholds in the decoupling criteria.

C

Systems and Control for Max-Plus Algebra

The issue of resolving a delay in a max-plus system is very much the same problem as steering the system from one state to another. This aim is therefore parallel to problems occurring in mathematical systems and control theory. In this appendix, we will take the time to reformulate some of the definitions of this report into the language of systems and control theory. Doing so will not only show the link between max-plus systems and conventional systems, but also reveal some additional interactions that may provide topics for further study.

The contents of this chapter are meant to situate max-plus algebra in the context of systems and control theory. None of the notation in this chapter are used in any other part of the report, as the report aims to build up an intuitive formulation of max-plus networks and extensions without relying on existing mathematics.

C.1. Max-Plus, Systems and Slow-Downs

We start by showing the similarity between max-plus systems and conventional systems. Throughout this report, we have assumed that we are aware of the exact position of each train. This is not an unrealistic assumption, but there are two reasons we may want to reconsider it.

- In some systems with less advanced equipment, we may not know the exact departure time of each station.
- Basing resolution sequences on the entire network causes very large computation times. Reducing the amount of departure times to consider in each time step, decreases this time.

We can add the restriction of only being able to observe some trains in the network as follows:

$$\begin{cases} \mathbf{x}(k+1) & = A \otimes \mathbf{x}(k) \\ \mathbf{y}(k+1) & = B \otimes \mathbf{x}(k) \end{cases}$$

Where B is a matrix containing some of the rows of A. When looking at the above system, we can clearly see its similarity to conventional systems.

The above system corresponds to a regular max-plus model. We now seek to extend this model to a switching model while still using system theoretical notations. We can do this, by adding the control term to the model, in the same fashion as conventional systems theory.

$$\begin{cases} \mathbf{x}(k+1) & = A \otimes \mathbf{x}(k) \oplus \mathbf{u}(k) \\ \mathbf{y}(k+1) & = B \otimes \mathbf{x}(k) \end{cases} \tag{C.1}$$

The question now becomes: how can we interpret this input. Since the input term is added to $A \otimes \mathbf{x}(k)$ in the max-plus sense, the next departure times are the maximum between the departure times in $A \otimes \mathbf{x}(k)$ and the departure times in $\mathbf{u}(k)$. This means that the input corresponds to increases in the departure times, which can be interpreted as slow downs of the trains. From this, one could conclude that the input term $\mathbf{u}(k)$ can be interpreted as the delay onset in time step k, but this is not logical, as we should be able to control the input term. This means that $\mathbf{u}(k)$ corresponds to slow downs of the trains in the network, chosen by the network operator.

Slowing down trains is one aspect of the switching model, but speeding up is the other. In the above system, the input term only allows slow-downs, so it seems we can not quite model switching perfectly. In the next section however, we will show that speed-ups can be modelled using these slow-downs, resulting in a properly implemented switching model.

C.2. Slowing Down instead of Speeding Up

In this report, we always assumed that the standard adjacency matrix, A_0, of an adjacency class was the matrix containing the standard commutes of the corresponding train networks. When trains sped up, the entries in this standard matrix would then be decreased to yield the matrix corresponding to the sped up commutes. Since we can not speed up the commutes in the system (C.1), we will assume the standard matrix A_0 to be the matrix containing all the fastest commutes. The input term $\mathbf{u}(k)$ can then be used to slow down the trains in case speed-ups are not necessary.

Example 26 *Let $\mathcal{M}_S$ be a switching max-plus model. The standard commute times are shown in the following adjacency matrix.*

$$A = \begin{pmatrix} 4 & 6 \\ 4 & 6 \end{pmatrix}$$

Both arcs with commute time 6 can be sped up by 1 time unit. In the regular switching model used in this report, the resulting adjacency class is

$$(A_0, A_1, A_2, A_3) = \left(\begin{pmatrix} 4 & 6 \\ 4 & 6 \end{pmatrix}, \begin{pmatrix} 4 & 5 \\ 4 & 6 \end{pmatrix}, \begin{pmatrix} 4 & 6 \\ 4 & 5 \end{pmatrix}, \begin{pmatrix} 4 & 5 \\ 4 & 5 \end{pmatrix} \right).$$

In the system theoretical model, there is only one adjacency matrix

$$A' = \begin{pmatrix} 4 & 5 \\ 4 & 5 \end{pmatrix}.$$

The input term $\mathbf{u}(k)$ can be set as the pre-determined timetable of the system, so trains will never depart before their scheduled time.

C.3. State Controller

In the previous section, we simply let the input $\mathbf{u}(k)$ be equal to the pre-determined time table, so trains do not depart too early. We however also implement the input similarly to the way it is often done in conventional systems theory, namely as a function of $\mathbf{x}(k)$:

$$\mathbf{u}(k) = C \otimes \mathbf{x}(k)$$

Since the max-plus multiplication is distributive with regards to the max-plus addition, we can now rewrite the system as

$$\begin{cases} \mathbf{x}(k+1) & = (A \oplus C) \otimes \mathbf{x}(k) \\ \mathbf{y}(k+1) & = B \otimes \mathbf{x}(k) \end{cases}$$

If we now let C be one of the adjacency matrices of the system, A_i, then we know that

$$A \oplus A_i = A_i \tag{C.2}$$

as A is the most sped-up adjacency matrix of the system. Using this method for implementing the input, we can return to our way of denoting the standard adjacency matrices we used in the regular switching system. We can now write every adjacency class of matrices as follows:

$$\mathcal{A} = (A', A_0, A_1, \dots, A_n)$$

which corresponds to the system

$$\begin{cases} \mathbf{x}(k+1) & = (A' \oplus A_i) \otimes \mathbf{x}(k) \\ \mathbf{y}(k+1) & = B \otimes \mathbf{x}(k) \end{cases}$$

and i is chosen at every time step. Because the notation for the standard model now matches the notation for the system theoretical model, we can apply the delay resolution methods to the system theoretical model as-is, without requiring any changes. This can be confirmed by the fact that as long as equality C.2 holds, we have that $(A' \oplus A_i) \otimes \mathbf{x}(k) = A_i \otimes \mathbf{x}(k)$, which is the same recurrence relation we used to formulate our resolution methods.

We note that it is important for the equality in equation C.2 to hold for the above input method to work. In the case where specific combinations of speed-ups are not possible, the equality does not hold, which may cause undesirable behaviour.

C.4. Delay Resolution and Control

As stated in the previous section, the delay resolution methods we formulated in this report can be applied to the system theoretical max-plus model, and as such, we will not further discuss them much. The subject of this section is the link between control theory and delay resolution. We will be talking about three common controls: State feedback, model predictive control and repetitive control.

State feedback is a control method where the input of a time step is partially determined using the state in that time step. This method is the one we used when formulating our input, $\mathbf{u}(k) = C \otimes \mathbf{x}(k)$. State feedback is a very useful tool in control theory, as using the current state to determine the control is very intuitive. A downside of feedback control however, is that it only takes into account the current state, not future or past states. Both these issues can be solved using the other two control methods we mentioned, predictive control and repetitive control.

In order to consider future stated, model predictive control (MPC) can be used. With MPC, we predict future perturbations from the expected course of events and base our input on these predicted perturbations (Schwenzer et al., 2021). An example of this is rush hour. During rush hour, the amount of delays may increase. To combat this increase, we could decide to more drastically resolve delays leading up to rush hour as to not be left with residual delays in addition to new delays.

The use of repetitive control (RC) fixes the issue of considering past states. Repetitive control is a method of control used on periodic signals (Ramos et al., 1970). Though the delays in our models are not periodic signals, we can use the underlying concept of RC to resolve recurring delays. If a certain delay or set of delays has already occurred and been resolved in the past, the same resolution sequence may once again work. Given enough data, delay resolution methods could thus use pre-optimised methods for resolving common delays or become better at resolving recurring delays.

Neither of the above control methods will be implemented, as they are massive concepts of their own. Both could be interesting subjects for further research, linking max-plus algebra and systems theory even further.

D
Python Code

This appendix contains all the python code used for the computations shown in this thesis. The code is split up into 3 major parts: Classes and functions, examples and the simulation. All three parts are imported by the `main.py` file found below.

```
from examples import *
from simulation import Simulation


def main():
    # Enter example here
    example = 26
    make_example(example)
    Sim = Simulation()

    #Uncomment to run a single simulation for various p
    p_range = 4
    Sim.simulation_various_p(p_range)

    # Uncomment to run 100 sample simulation
    p_range = 2
    Sim.stat_run_sim(p_range)
    return

if __name__ == '__main__':
    main()
```

D.1. Classes and Functions

This section contains the classes and functions that acted as the back-bone of the computations.

D.1.1. Classes

```
# Packages
import numpy as np
import time
import random
import copy
import math

# Modules
from functions import *

eps = '\u03B5'
inf = 'infty'
tab_len = 4
random.seed(123)


```

```python
# All important max-plus operations
class MaxPlus:
    class Operation:
        @staticmethod
        def max(lst):
            M = eps
            for el in lst:
                if el == eps:
                    continue
                elif M == eps:
                    M = el
                elif el > M:
                    M = el
            return M

        @staticmethod
        def min(lst):
            m = inf
            for el in lst:
                if el == inf:
                    continue
                elif m == inf:
                    m = el
                elif el == eps:
                    return eps
                elif el < m:
                    m = el
            return m

        @staticmethod
        def add(a, b):
            if a == eps or b == eps:
                return eps
            return a + b

        @staticmethod
        def mul(a, b):
            if a == eps or b == eps:
                return eps
            return a * b

        @staticmethod
        def state_add(x1, x2):
            nx = []
            for i, el in enumerate(x1):
                nx.append(MaxPlus.Operation.add(el, x2[i]))
            return nx

        @staticmethod
        def state_max(state_lst):
            nx = []
            for i in range(len(state_lst[0])):
                row = []
                for j in range(len(state_lst)):
                    row.append(state_lst[j][i])
                nx.append(MaxPlus.Operation.max(row))

            return nx

        @staticmethod
        def scalar_add(x, c):
            nx = []
            for el in x:
                nx.append(MaxPlus.Operation.add(el, c))
            return nx

        @staticmethod
        def scalar_mul(x, c):
            nx = []
            for el in x:
                nx.append(MaxPlus.Operation.mul(el, c))
```

```
            return nx

    @staticmethod
    def base(x):
        if type(x[0]) != list:
            sub = MaxPlus.Operation.min(x)
            return MaxPlus.Operation.scalar_add(x, -sub)
        res = []
        for state in x:
            res.append(MaxPlus.Operation.base(state))
        return res

    @staticmethod
    def mat_mul(A, B):
        new_mat = []
        for i in range(len(A)):
            row = []
            for j in range(len(A[i])):
                entry = []
                for k in range(len(A[i])):
                    entry.append(MaxPlus.Operation.add(A[i, k], B[k, j]))
                row.append(MaxPlus.Operation.max(entry))
            new_mat.append(row)
        return new_mat

    @staticmethod
    def norm1(A, B):
        res = 0
        for i in range(len(A)):
            if not type(A[i]) == int and not type(A[i]) == float:
                for j in range(len(A[i])):
                    res += abs(A[i][j] - B[i][j])
            else:
                res += abs(A[i] - B[i])
        return res

    @staticmethod
    def sequence_norm1(Alst, J1, J2):
        res = 0
        for i in range(min(len(J1), len(J2))):
            res += MaxPlus.Operation.norm1(Alst[J1[i]], Alst[J2[i]])
        return res

    @staticmethod
    def normmax(A, B):
        res = []
        for i in range(len(A)):
            if not type(A[i]) == int and not type(A[i]) == float:
                for j in range(len(A[i])):
                    res.append(abs(A[i][j] - B[i][j]))
            else:
                res.append(abs(A[i] - B[i]))
        return MaxPlus.Operation.max(res)

    @staticmethod
    def score1(xd, x):
        return [MaxPlus.Operation.normmax(xd, x), MaxPlus.Operation.norm1(xd, x)]

    @staticmethod
    def argmin_score(score_lst):
        ret = [1 for _ in score_lst]
        for i in range(len(score_lst[0])):
            entry_lst = []
            for j, score in enumerate(score_lst):
                if ret[j] == 1:
                    entry_lst.append(score[i])
            entry_min = min(entry_lst)
            for j, score in enumerate(score_lst):
                if score[i] > entry_min:
                    ret[j] = 0
        out = []
```

```
            for i in range(len(ret)):
                if ret[i]:
                    out.append(i)
            return out

        @staticmethod
        def station_delay(A, station, delay):
            A = copy.deepcopy(A)
            for i in range(len(A)):
                A[i][station] = MaxPlus.Operation.add(A[i][station], delay)
            return A

        @staticmethod
        def track_delay(A, arc, delay):
            A = copy.deepcopy(A)
            i, j = arc[0], arc[1]
            A[i][j] = MaxPlus.Operation.add(A[i][j], delay)
            return A

    class Regime:
        @staticmethod
        def max_plus(x, A):
            new_x = []
            for i in range(len(x)):
                entry = []
                for j in range(len(x)):
                    entry.append(MaxPlus.Operation.add(x[j], A[i][j]))
                new_x.append(MaxPlus.Operation.max(entry))
            return new_x

        @staticmethod
        def regime(x, A, n):
            res_lst = [x]
            for _ in range(n):
                x = MaxPlus.Regime.max_plus(x, A)
                res_lst.append(x)
            return res_lst

        @staticmethod
        def switching_regime(x, Alst, J):
            res_lst = [x.copy()]
            for ind in J:
                x = MaxPlus.Regime.max_plus(x, Alst[ind])
                res_lst.append(x)
            return res_lst

        @staticmethod
        def multi_switching_regime(x, class_list, class_index, J):
            res_lst = [x.copy()]
            for i in range(min(len(class_index), len(J))):
                adj_class = class_list[class_index[i]]
                index = J[i]
                A = adj_class[index]
                x = MaxPlus.Regime.max_plus(x, A)
                res_lst.append(x)
            return res_lst

        @staticmethod
        def delay_sequence(regime, delayed_regime):
            l = min(len(regime), len(delayed_regime))
            res = []
            for i in range(l):
                res.append(list_sub(regime[i], delayed_regime[i]))
            return res

    class Simulation:
        def __init__(self, x0, Omega, H, J, stoch, end, method=None, method_param=0, decouple
    =False, info=False):
            self.x = x0
            self.regime = [self.x]
            self.Omega = Omega
```

```
        self.H = H
        self.eqJ = J
        self.J = []
        if not method:
            self.J = self.eqJ
        self.info = info

        self.method = method
        self.method_param = method_param

        self.TD_dict = {}
        self.SD_dict = {}
        self.TF_lst = []

        self.decouple = decouple
        if not decouple:
            self.decouple = []
        self.decouple_lst = []
        self.decoupled = 0

        self.end = end
        self.equilibrium = MaxPlus.Regime.multi_switching_regime(x0, Omega, H, J)
        self.delays = [[0 for _ in x0]]
        self.total_delay = [0]

        self.onset_stoch = stoch[0]
        self.sustain_stoch = stoch[1]
        self.delay_distr = stoch[2]

        self.dim = len(Omega[0][0])
        self.arcs = self.get_arcs()

        self.current = None
        self.time_step = 0

        self.time = time.time()

    def simulate(self):
        while self.time_step < self.end:
            self.current = copy.deepcopy(self.Omega[self.H[self.time_step]])
            self.step()
            if self.info:
                self.print_info()
            self.SD_dict = {}
            self.time_step += 1

    def step(self):
        self.decouple_lst = []
        self.sustain_delay()
        self.switch()
        self.current = self.current[self.J[-1]]
        expected_departure = MaxPlus.Operation.scalar_mul(MaxPlus.Regime.max_plus(self.x,
 self.current), -1)
        self.onset_delay()
        self.apply_delay()
        if self.time_step in self.decouple:
            self.shift_decouple()
        self.decouple_correction()
        self.x = MaxPlus.Regime.max_plus(self.x, self.current)
        self.regime.append(self.x)
        self.delays.append(MaxPlus.Operation.state_add(self.x, expected_departure))


    def switch(self):
        if self.method == 'Combinatorial':
            res = MaxPlus.Solve.multi_combinatorial(self.Omega, self.H, self.equilibrium[
self.time_step:],
                                                    self.x, timer=False, timeout=30)

        elif self.method == 'P-Composite':
            p = self.method_param
```

```
                ts = self.time_step
                # res = MaxPlus.Solve.multi_p_greedy(self.Omega, self.H[ts:], self.equilibrium[ts:],
                #                                    self.x, p, timer=False, timeout=30)

                res, _ = MaxPlus.Solve.multi_partial_comb(self.Omega, self.H[ts:ts+p], self.equilibrium[ts:ts+p+1],
                                                           self.x)

            else:
                return
            if not res:
                index = self.eqJ[self.time_step]
            else:
                index = res[0]
            self.J.append(index)

    def onset_delay(self):
        for outbound in range(self.dim):
            onset_SD = Probability.bernoulli(self.onset_stoch['SD'])
            if onset_SD:
                delay = Probability.discrete(self.delay_distr['SD'])
                self.SD_dict[outbound] = delay

        for arc in self.arcs:
            onset_TD = Probability.bernoulli(self.onset_stoch['TD'])
            if onset_TD:
                if arc not in self.TD_dict:
                    self.TD_dict[arc] = Probability.discrete(self.delay_distr['TD'])
                else:
                    self.TD_dict[arc] = max(self.TD_dict[arc], Probability.discrete(self.delay_distr['TD']))

            onset_TF = Probability.bernoulli(self.onset_stoch['TF'])
            if onset_TF and arc not in self.TF_lst:
                self.TF_lst.append(arc)

    def sustain_delay(self):
        dict_copy = self.TD_dict.copy()

        for arc in dict_copy:
            sustain = Probability.bernoulli(self.sustain_stoch['TD'])
            if not sustain:
                del(self.TD_dict[arc])

        for arc in self.TF_lst:
            sustain = Probability.bernoulli(self.sustain_stoch['TF'])
            if not sustain:
                self.TF_lst.remove(arc)

        return

    def apply_delay(self):
        for arc in self.TD_dict:
            track = self.arcs[arc]
            self.current = MaxPlus.Operation.track_delay(self.current, track, self.TD_dict[arc])

        for arc in self.TF_lst:
            track = self.arcs[arc]
            self.current = MaxPlus.Operation.track_delay(self.current, track, eps)

        for outbound in self.SD_dict:
            self.current = MaxPlus.Operation.station_delay(self.current, outbound, self.SD_dict[outbound])

    def shift_decouple(self):
        for j in range(len(self.x)):
            for i in range(len(self.current)):
                if self.current[i][j] != eps:
                    arrival = self.x[j] + self.current[i][j]
```

```
                    if arrival >= self.equilibrium[self.time_step+1][j] + self.current[i][j]//2:
                        self.current = MaxPlus.Operation.track_delay(self.current, [i,j], eps)
                        self.decouple_lst.append(i+self.dim*j)
                        self.decoupled += 1



    def decouple_correction(self):
        for arc in self.TF_lst+self.decouple_lst:
            track = self.arcs[arc]
            inbound = track[0]
            outbound = track[1]
            self.current[inbound][outbound] = self.equilibrium[self.time_step+1][inbound] - self.x[outbound]

    def get_arcs(self):
        arcs = {}
        for j in range(self.dim):
            for i in range(self.dim):
                A = self.Omega[0][0]
                if A[i][j] != eps:
                    arcs[i+self.dim*j] = [i,j]
        return arcs

    def print_info(self):
        print(f'Time Step: {self.time_step}')
        print('Track delays:')

        for delay in self.TD_dict:
            print(f'\t{self.arcs[delay]}: {self.TD_dict[delay]}')

        print('Station delays:')

        for delay in self.SD_dict:
            print(f'\t{delay}: {self.SD_dict[delay]}')

        print('Track Failures:')

        for delay in self.TF_lst:
            print(f'\t{self.arcs[delay]}')
        print(f'Decouple: {self.decouple_lst}')
        print(f'Time: {time.time()-self.time}')
        print(200*'#')

    # @staticmethod
    # def get_delay_index(A, p_lst):
    #     delayed = []
    #     station_delayed = []
    #     decoupled = []
    #     for i in range(len(A)):
    #         for j in range(len(A)):
    #             # Check for track delay
    #             if Probability.bernoulli(p_lst[0]):
    #                 delayed.append([i, j])
    #             # Check for track failure
    #             if Probability.bernoulli(p_lst[2]):
    #                 decoupled.append([i, j])
    #         # Check for station delay
    #         if Probability.bernoulli(p_lst[1]):
    #             station_delayed.append(i)
    #
    #     return delayed, station_delayed, decoupled
    #
    # @staticmethod
    # def stoch_delays(A, index, delay_lst, p_lst):
    #     A = A.copy()
    #     delay = Probability.discrete(delay_lst, p_lst)
    #     i, j = index[0], index[1]
    #     A[i][j] = MaxPlus.Operation.add(A[i][j], delay)
```

```
    #     return A
    #
    # @staticmethod
    # def station_delay(A, station, delay_lst, p_lst):
    #     A = A.copy()
    #     delay = Probability.discrete(delay_lst, p_lst)
    #     for i in range(len(A)):
    #         A[i][station] = MaxPlus.Operation.add(A[i][station], delay)
    #     return A
    #
    # @staticmethod
    # def get_delayed_matrix(A, bernoulli_p_lst, delay_lst, delay_p_lst, station_delay_lst, station_delay_p_lst):
    #     delayed, station_delayed, decoupled = MaxPlus.Operation.get_delay_index(A, bernoulli_p_lst)
    #     for index in delayed:
    #         A = MaxPlus.Operation.stoch_delay(A, index, delay_lst, delay_p_lst)
    #
    #     for station in station_delayed:
    #         A = MaxPlus.Operation.station_delay(A, station, station_delay_lst, station_delay_p_lst)
    #
    #     for index in decoupled:
    #         A = MaxPlus.Operation.stoch_delay(A, index, [eps], [1])
    #     return A
    #
    # @staticmethod
    # def stochastic_regime(x: list, omega: list, H: list, J: list, stoch):
    #     TD_dict = {}
    #     TF_lst = []
    #
    #     for

class Solve:
    @staticmethod
    def karp(A):
        dim = len(A)
        x = [eps for _ in range(dim)]
        x[0] = 0
        regime = MaxPlus.Regime.regime(x, A, dim)

        meta_lst = []
        for k in range(dim):
            xn = regime[dim]
            xk = regime[k]

            xk_min = MaxPlus.Operation.scalar_mul(xk, -1)

            meta_lst.append(MaxPlus.Operation.scalar_mul(MaxPlus.Operation.state_add(xn, xk_min), 1 / (dim - k)))

        max_lst = []
        for arr in meta_lst:
            max_lst.append(MaxPlus.Operation.max(arr))

        return MaxPlus.Operation.min(max_lst)

    @staticmethod
    def power(A):
        dim = len(A)
        x = [0 for _ in range(dim)]
        done = False
        regime = [x]
        while not done:
            x = MaxPlus.Regime.max_plus(x, A)
            for k, state in enumerate(regime):
                for i in range(1, MaxPlus.Operation.max(x) + 1):
                    translate = True
                    for j in range(len(state)):
                        if state[j] + i != x[j]:
                            translate = False
```

```
                        if translate == True:
                            done = True
                            p = len(regime)
                            q = k
                            c = i
                regime.append(x)

            mu = c / (p - q)
            ev_lst = []
            for j in range(1, (p - q) + 1):
                el = MaxPlus.Operation.scalar_add(regime[q + j - 1], mu * (p - q - j))
                ev_lst.append(el)

            ev = MaxPlus.Operation.state_max(ev_lst)
            ev = MaxPlus.Operation.scalar_add(ev, -MaxPlus.Operation.min(ev))

            return mu, ev

        @staticmethod
        def combinatorial_method(Alst, regime, x0, minimal=False, J_ref=None, timer=False,
    timeout = 30):
            st = time.time()
            res = []
            for i in range(len(regime)):
                res = []
                exhaust = comb([j for j in range(len(Alst))], i)
                for J in exhaust:
                    x = x0.copy()
                    for j in J:
                        x = MaxPlus.Regime.max_plus(x, Alst[j])
                    if x == regime[i]:
                        res.append(J)
                    if time.time()-st > timeout:
                        print(f'Runtime Error: Iteration {i} reached')
                        return i
                if timer:
                    print(f'Iteration {i}: {time.time() - st}')
                if res:
                    break

            out = res
            if minimal:
                min_norm = []
                for R in res:
                    Anorm = sum([MaxPlus.Operation.norm1(Alst[R[i]], Alst[J_ref[i]]) for i in
    range(len(R))])
                    min_norm.append(Anorm)
                min_norm = MaxPlus.Operation.min(min_norm)

                out = []

                for R in res:
                    Anorm = sum([MaxPlus.Operation.norm1(Alst[R[i]], Alst[J_ref[i]]) for i in
    range(len(R))])
                    if Anorm == min_norm:
                        out.append(R)
            return out

        @staticmethod
        def greedy_method(Alst, regime, x0, minimal=False, J=None):
            x = x0
            res = []
            for i in range(len(regime) - 1):
                score_lst = []
                for j in range(len(Alst)):
                    nx = MaxPlus.Regime.max_plus(x, Alst[j])
                    score = MaxPlus.Operation.score1(nx, regime[i+1])
                    if minimal:
                        score.append(MaxPlus.Operation.norm1(Alst[j], Alst[J[i]]))
                    score_lst.append(score)
```

```python
            min_score = MaxPlus.Operation.argmin_score(score_lst)
            index = min_score[0]

            x = MaxPlus.Regime.max_plus(x, Alst[index])
            res.append(index)
        return res

    @staticmethod
    def partial_combinatorial_method(Alst, regime, x0, minimal=False, Jref=None):
        exhaust = comb([j for j in range(len(Alst))], len(regime) - 1)
        score_lst = []
        for J in exhaust:
            x = x0.copy()
            Anorm = 0
            for i, j in enumerate(J):
                x = MaxPlus.Regime.max_plus(x, Alst[j])
                if minimal:
                    Anorm += MaxPlus.Operation.norm1(Alst[j], Alst[Jref[i]])

            score = MaxPlus.Operation.score1(x, regime[-1])
            if minimal:
                score.append(Anorm)

            score_lst.append(score)

        index = MaxPlus.Operation.argmin_score(score_lst)[0]
        res = exhaust[index]
        nx = MaxPlus.Regime.switching_regime(x0.copy(), Alst, res)[-1]
        if nx == regime[-1]:
            return MaxPlus.Solve.combinatorial_method(Alst, regime, x0)[0], nx

        return res, nx

    @staticmethod
    def p_greedy(Alst, regime, x0, p, minimal=False, J=None, timer=False, timeout=30):
        st = time.time()
        n = len(regime) // p
        x = x0.copy()
        res = []
        for i in range(n):
            regime_part = regime[i * p:(i + 1) * p + 1]
            if minimal:
                J_part = J[i * p:(i + 1) * p + 1]
            else:
                J_part = None
            R_part, x = MaxPlus.Solve.partial_combinatorial_method(Alst, regime_part, x,
 minimal=minimal, Jref=J_part)
            res += R_part
            if len(R_part) != p or x == regime_part[-1]:
                return res
            if timer:
                print(f'Iteration {i}: {time.time() - st}')
            if time.time()-st > timeout:
                print(f'Runtime Error: Iteration {i} reached')
        return res

    @staticmethod
    def multi_combinatorial(array, H, regime, x0, minimal=False, J_ref=None, timer=False,
 timeout=30):
        st = time.time()
        res = []
        for i in range(len(regime)):
            res = []
            exhaust = multi_comb([[j for j in range(len(array[k]))] for k in range(len(
array))], H[:i])
            for J in exhaust:
                x = x0.copy()
                for k, j in enumerate(J):
                    x = MaxPlus.Regime.max_plus(x, array[H[k]][j])
                if x == regime[i]:
                    res.append(J)
```

```
                if time.time() - st > timeout:
                    print(f'Runtime Error: Iteration {i} reached')
                    return i
            if timer:
                print(f'Iteration {i}: {time.time() - st}')
            if res:
                break


        out = res
        if minimal:
            min_norm = []
            for R in res:
                Anorm = sum([MaxPlus.Operation.norm1(array[H[i]][R[i]], array[H[i]][J_ref[i]]) for i in range(len(R))])
                min_norm.append(Anorm)
            min_norm = MaxPlus.Operation.min(min_norm)

            out = []

            for R in res:
                Anorm = sum([MaxPlus.Operation.norm1(array[H[i]][R[i]], array[H[i]][J_ref[i]]) for i in range(len(R))])
                if Anorm == min_norm:
                    out.append(R)
        return out

    @staticmethod
    def multi_partial_comb(array, H, regime, x0, minimal=False, Jref=None):
        exhaust = multi_ncomb(array, H)
        score_lst = []
        for J in exhaust:
            x = x0.copy()
            Anorm = 0
            for i, j in enumerate(J):
                Alst = array[H[i]]
                x = MaxPlus.Regime.max_plus(x, Alst[j])
                if minimal:
                    Anorm += MaxPlus.Operation.norm1(Alst[j], Alst[Jref[i]])
            score = MaxPlus.Operation.score1(x, regime[-1])

            if minimal:
                score.append(Anorm)

            score_lst.append(score)

        index = MaxPlus.Operation.argmin_score(score_lst)[0]
        res = exhaust[index]
        nx = MaxPlus.Regime.multi_switching_regime(x0.copy(), array, H, res)[-1]
        if nx == regime[-1]:
            return MaxPlus.Solve.multi_combinatorial(array, H, regime, x0)[0], nx

        return res, nx

    @staticmethod
    def multi_p_greedy(array, H, regime, x0, p, minimal=False, J=None, timer=False, timeout=30):
        res = []
        l = min(len(H), len(regime)) // p
        x = x0.copy()
        st = time.time()
        for i in range(l):
            regime_part = regime[i*p: i*p+p+1]
            H_part = H[i * p:(i + 1) * p]
            if minimal:
                J_part = J[i * p:(i + 1) * p]

            else:
                J_part = None
            R_part, x = MaxPlus.Solve.multi_partial_comb(array, H_part, regime_part, x, minimal=minimal,
```

```
                                                            Jref=J_part)
                res += R_part
                if len(R_part) != p or x == regime_part[-1]:
                    return res
                if timer:
                    print(f'Iteration {i}: {time.time() - st}')
                if time.time() - st > timeout:
                    print(f'Runtime Error: Iteration {i} reached')
                    return
            return res

    class String:

        @staticmethod
        def regime_string(regime, integer=False):
            string = ''
            for i in range(len(regime[0])):
                row = f'{i}:\t'
                for j in range(len(regime)):
                    time = regime[j][i]
                    if integer:
                        time = int(time)
                    row += f'{time}\t\t'
                string += f'{row}\n'
            return string[:-1]

        @staticmethod
        def mat_string(A):
            string = ''
            for i in range(len(A)):
                row = f'[\t'
                for j in range(len(A[i])):
                    row += f'{A[i][j]}\t'
                string += f'{row}]\n'
            return string[:-1]

        @staticmethod
        def multi_matrix(names, mats):
            string = ''
            nrows = len(mats[0])
            nmats = len(mats)
            for i in range(nrows):
                row = ''
                for j in range(nmats):
                    name = names[j]
                    matrix = mats[j]
                    if i == nrows//2 - 1:
                        row += f'{name} = \t[\t'
                    else:
                        row += (len(name)//tab_len + 2) * '\t' + '[\t'

                    if type(matrix[i]) != list:
                        row += f'{matrix[i]}\t'
                    else:
                        for k in range(len(matrix)):
                            row +=  f'{matrix[i][k]}\t'

                    row += ']\t'
                row += '\n'
                string += row
            return string[:-2]


class Probability:
    @staticmethod
    def bernoulli(p):
        num = random.randint(0, 99)
        if num < p*100:
            return True
        return False
```

```
    @staticmethod
    def discrete(distr):
        event_lst = []
        p_lst = []
        for event in distr:
            event_lst.append(event)
            p_lst.append(distr[event])
        P = sum(p_lst)
        num = random.randint(0, P-1)
        intervals = cumulative_interval(p_lst)
        for i in range(len(p_lst)):
            if intervals[i] <= num < intervals[i + 1]:
                return event_lst[i]
        print('error')


class Statistics:
    @staticmethod
    def list_dif(lst1, lst2):
        res = 0
        for i in range(len(lst1)):
            res += abs(lst1[i]-lst2[i])
        return res

    @staticmethod
    def mean(lst):
        return sum(lst)/len(lst)

    @staticmethod
    def SD(lst):
        S = 0
        for el in lst:
            S += (el-Statistics.mean(lst))**2
        Var = S / (len(lst)-1)
        return math.sqrt(Var)


class Stochast:
    def __init__(self):
        self.dict = {}

    def add_bernoulli(self, p, label):

        def bernoulli():
            P = 100 * p
            num = random.randint(0, 100)
            if num <= P:
                return True
            return False

        self.dict[label] = bernoulli

    def add_discr_distr(self, p_lst, event_lst, label):

        def discr():
            P = sum(p_lst)
            num = random.randint(0, P)
            intervals = cumulative_interval(p_lst)
            for i in range(len(p_lst)):
                if intervals[i] <= num < intervals[i+1]:
                    return event_lst[i]

        self.dict[label] = discr
```

D.1.2. Functions

```
eps = '\u03B5'

def fact(n):
    if n == 0:
```

```
        return 1
    return fact(n-1)

def ncomb(n,k):
    num = fact(n)
    denom = fact(k)*fact(n-k)
    return num/denom

def comb(lst, n):
    if n == 0:
        return [[]]
    else:
        pre_lst =  comb(lst, n-1)
    new_lst = []
    for part in pre_lst:
        for el in lst:
            copy_part  =part.copy()
            copy_part.append(el)
            new_lst.append(copy_part)
    return new_lst

def multi_comb(lst, H):
    n = len(H)
    if n == 0:
        return [[]]
    res = []
    old_res = multi_comb(lst, H[:-1])
    for el in lst[H[-1]]:
        for lst in old_res:
            nlst = lst.copy()
            nlst.append(el)
            res.append(nlst)
    return res

def multi_ncomb(lst, H):
    n = len(H)
    if n == 0:
        return [[]]
    res = []
    old_res = multi_ncomb(lst, H[:-1])
    for i, el in enumerate(lst[H[-1]]):
        for lst in old_res:
            nlst = lst.copy()
            nlst.append(i)
            res.append(nlst)
    return res

def argmin(lst):
    min_lst = min(lst)
    res = []
    for i, el in enumerate(lst):
        if el == min_lst:
            res.append(i)
    return res

def list_min(lst_lst, ind=0):
    res = []
    el_lst = []
    for lst in lst_lst:
        el_lst.append(lst[ind])

    lst_min = min(el_lst)
    for lst in lst_lst:
        if lst[ind] == lst_min:
            res.append(lst)
    return res

def cumulative_interval(lst):
    res = [0]
    current = 0
    for num in lst:
```

```
            current += num
            res.append(current)
        return res

def list_sub(lst1, lst2):
    if len(lst1) != len(lst2):
        return
    res = []
    for i in range(len(lst1)):
        res.append(lst1[i] - lst2[i])
    return res

def count(lst, el):
    c = 0
    for i in lst:
        if i == el:
            c += 1
    return c

def get_arcs(A):
    arcs = []
    for i in range(len(A)):
        for j in range(len(A)):
            if A[i][j] != eps:
                arcs.append([i,j])
    return arcs
```

D.2. Examples

This section contains the code for the examples used throughout the report. All examples can be run through the `main.py` file.

```
import numpy as np

from classes import MaxPlus as MP
from classes import eps, inf


def make_example(n):
    k = 150
    print(k * '=')
    print(k * '=')

    print(f'Example {n}:\n')
    eval(f'example_{n}()')

    print(k * '=')
    print(k * '=')


def example_1():
    A = [[2, 5],
         [3, 3]]

    x0 = [0, 0]

    print(MP.String.multi_matrix(['x0', 'A'], [x0, A]))

    print('\nResult: \n')

    regime = MP.Regime.regime(x0, A, 10)
    print(MP.String.regime_string(regime))


def example_3():
    A = ([[2, eps, eps, 2],
          [3, eps, 4, 4],
          [eps, 2, 3, eps],
          [1, eps, 1, eps]])

```

```
    x0 = [0, 0, 0, 0]

    print(MP.String.multi_matrix(['x0', 'A'], [x0, A]))

    print('\nResult: \n')

    regime = MP.Regime.regime(x0, A, 10)
    print(MP.String.regime_string(regime))


def example_4():
    A = [[2, 5], [3, 3]]
    x0 = [2, 0]

    print(MP.String.multi_matrix(['x0', 'A'], [x0, A]))

    print('\nResult: \n')

    regime = MP.Regime.regime(x0, A, 10)
    base_regime = MP.Operation.base(regime)

    print('Departure sequence:')
    print(MP.String.regime_string(regime))
    print('\nBase Sequence:')
    print(MP.String.regime_string(base_regime))


def example_5():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 4], [3, 3]]
    A2 = [[2, 5], [3, 2]]
    A3 = [[2, 4], [3, 2]]
    Alst = [A0, A1, A2, A3]
    J = [0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0]

    x0 = [1, 0]

    print(MP.String.multi_matrix(['x0', 'A0', 'A1', 'A2', 'A3'], [x0, A0, A1, A2, A3]))
    print(f'\nJ = {J}')

    print('\nResult: \n')

    regime = MP.Regime.switching_regime(x0, Alst, J)
    base_regime = MP.Operation.base(regime)

    print('Departure sequence:')
    print(MP.String.regime_string(regime))
    print('\nBase Sequence:')
    print(MP.String.regime_string(base_regime))
    print('\nEigenvalue and eigenvector:')
    print(MP.Solve.power(A0))
    print(MP.Solve.power(A3))


def example_6():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 4], [3, 3]]
    A2 = [[2, 5], [3, 2]]
    A3 = [[2, 4], [3, 2]]
    Alst = [A0, A1, A2, A3]

    x0 = [5, 4]
    xd = [7, 4]

    J = [0 for i in range(10)]

    print(MP.String.multi_matrix(['x0', 'A0', 'A1', 'A2', 'A3', 'xd'], [x0, A0, A1, A2, A3, xd]))
    print(f'\nJ = {J}')

    print('\nResult: \n')
```

```
    regime = MP.Regime.regime(x0, A0, 10)

    Jlst = MP.Solve.combinatorial_method(Alst, regime, xd)

    print('Departure sequence:')
    print(MP.String.regime_string(regime))

    for i, J_i in enumerate(Jlst):
        delayed_regime = MP.Regime.switching_regime(xd, Alst, J_i)
        print(f'\nDelay Solution {i+1}: J tilde = {J_i}')
        print(MP.String.regime_string(delayed_regime))


def example_8():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 5], [2, 3]]
    A2 = [[2, 6], [3, 3]]
    A3 = [[2, 6], [2, 3]]
    Alst = [A0, A1, A2, A3]

    x0 = [1, 0]

    J = [0, 0, 0, 0, 0, 0, 0, 0, 0]
    J_changed = [0, 0, 1, 2, 0, 0, 0, 0, 0]

    print(MP.String.multi_matrix(['x0', 'A0', 'A1', 'A2', 'A3'], [x0, A0, A1, A2, A3]))

    print('\nResult: \n')

    regime = MP.Regime.regime(x0, A0, len(J))
    changed_regime = MP.Regime.switching_regime(x0, Alst, J_changed)

    print(f'Departure sequence: J = {J}')
    print(MP.String.regime_string(regime))

    print(f'\nAltered sequence: J = {J_changed}')
    print(MP.String.regime_string(changed_regime))


def example_9():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 4], [3, 3]]
    A2 = [[2, 5], [3, 2]]
    A3 = [[2, 4], [3, 2]]
    Alst = [A0, A1, A2, A3]

    x0 = [5, 4]
    xd = [7, 4]

    J = [0 for i in range(10)]

    print(MP.String.multi_matrix(['x0', 'A0', 'A1', 'A2', 'A3', 'xd'], [x0, A0, A1, A2, A3,
     xd]))
    print(f'\nJ = {J}')

    print('\nResult:')

    regime = MP.Regime.regime(x0, A0, 10)
    Jlst = MP.Solve.combinatorial_method(Alst, regime, xd)

    for i, J_i in enumerate(Jlst):
        delayed_regime = MP.Regime.switching_regime(xd, Alst, J_i)
        print(f'\nDelay Solution {i + 1}: J tilde = {J_i}, ')
        print(MP.String.regime_string(delayed_regime))


def example_10():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 4], [3, 3]]
    A2 = [[2, 5], [3, 2]]
```

```
    A3 = [[2, 4], [3, 2]]
    Alst = [A0, A1, A2, A3]
    J = [0 for i in range(10)]

    print(MP.String.multi_matrix(['A0', 'A1', 'A2', 'A3'], [A0, A1, A2, A3]))

    print('\nResult:')

    x0 = [10, 10]
    xd = [11, 11]

    print('Solutions for:')
    print(MP.String.multi_matrix(['x0', 'xd'], [x0, xd]))

    regime = MP.Regime.regime(x0, A0, 10)

    J1 = MP.Solve.combinatorial_method(Alst, regime, xd, minimal=True, J_ref = J)[0]
    delayed_regime = MP.Regime.switching_regime(xd, Alst, J1)
    print(f'\nSolution: J tilde = {J1}')
    print(MP.String.regime_string(delayed_regime))

    x0 = [0, 0]
    xd = [1, 1]

    print('\nSolutions for:')
    print(MP.String.multi_matrix(['x0', 'xd'], [x0, xd]))

    regime = MP.Regime.regime(x0, A0, 10)

    J1 = MP.Solve.combinatorial_method(Alst, regime, xd, minimal=True, J_ref=J)[0]
    delayed_regime = MP.Regime.switching_regime(xd, Alst, J1)
    print(f'\nSolution: J tilde = {J1}')
    print(MP.String.regime_string(delayed_regime))


def example_11():
    size_A = 32
    m = 10
    TC = size_A**m
    print(f'Time Complexity: C*{TC}')
    print(f'Log Time Complexity: Log(C)+{np.log10(TC)}')


def example_12():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 4], [3, 3]]
    A2 = [[2, 5], [3, 2]]
    A3 = [[2, 4], [3, 2]]
    Alst = [A0, A1, A2, A3]
    x0 = [1,0]
    J = [0 for i in range(10)]
    matrices = ['A0', 'A1', 'A2', 'A3']

    xd = [3, 0]

    print(MP.String.multi_matrix(['x0', 'A0', 'A1', 'A2', 'A3', 'xd'], [x0, A0, A1, A2, A3,
    xd]))

    regime = MP.Regime.switching_regime(x0, Alst, J)

    print('\nDeparture Sequence:')
    print(MP.String.regime_string(regime))

    print('\nResult:')

    print('\nTime step 1:')
    x1lst = []
    names1 = []
    for i, A in enumerate(Alst):
        x1lst.append(MP.Regime.max_plus(xd, A))
        names1.append(f'{matrices[i]} X x0')
```

```
    print(MP.String.multi_matrix(names1, x1lst))

    xd = [5, 6]

    print('\nTime step 2:')
    x2lst = []
    names2 = []
    for i, A in enumerate(Alst):
        x2lst.append(MP.Regime.max_plus(xd, A))
        names2.append(f'{matrices[i]} X x1')

    print(MP.String.multi_matrix(names2, x2lst))

    xd = [10, 8]

    print('\nTime step 3:')
    x3lst = []
    names3 = []
    for i, A in enumerate(Alst):
        x3lst.append(MP.Regime.max_plus(xd, A))
        names3.append(f'{matrices[i]} X x2')

    print(MP.String.multi_matrix(names3, x3lst))

    xd = [13, 13]

    print('\nTime step 4:')
    x4lst = []
    names4 = []
    for i, A in enumerate(Alst):
        x4lst.append(MP.Regime.max_plus(xd, A))
        names4.append(f'{matrices[i]} X x3')

    print(MP.String.multi_matrix(names4, x4lst))


    xd = [3, 0]
    res = MP.Solve.greedy_method(Alst, regime, xd, minimal=True, J=J)
    print(f'\nMinimal Solution: J = {res}')
    delayed_regime = MP.Regime.switching_regime(xd, Alst, res)
    print(MP.String.regime_string(delayed_regime))


def example_13():
    A0 = [[eps, 5], [5, eps]]
    A1 = [[eps, 3], [3, eps]]
    A2 = [[eps, 2], [2, eps]]
    Alst = [A0, A1, A2]

    x0 = [0, 0]
    xd = [4, 4]

    J_greedy = [0 for i in range(10)]
    regime = MP.Regime.switching_regime(x0, Alst, J_greedy)

    names = ['x0', 'A0', 'A1', 'A2', 'xd']
    matrices = [x0, A0, A1, A2, xd]
    print(MP.String.multi_matrix(names, matrices))
    print('\nDeparture Sequence:')
    print(MP.String.regime_string(regime))

    print('\nResult:')

    J_greedy = MP.Solve.greedy_method(Alst, regime, xd)
    J_comb = MP.Solve.combinatorial_method(Alst, regime, xd)[0]

    regime_greedy = MP.Regime.switching_regime(xd, Alst, J_greedy)
    regime_comb = MP.Regime.switching_regime(xd, Alst, J_comb)

    print(f'\n(False) Solution with Greedy Method: J = {J_greedy}')
```

```
    print(MP.String.regime_string(regime_greedy))

    print(f'\nSolution with Combinatorial Method: J = {J_comb}')
    print(MP.String.regime_string(regime_comb))


def example_14():
    A0 = [[5, eps], [eps, 5]]
    A1 = [[3, eps], [eps, 3]]
    A2 = [[2, eps], [eps, 2]]
    Alst = [A0, A1, A2]

    x0 = [0, 0]
    xd = [100, 100]
    J = [0 for i in range(34)]
    regime = MP.Regime.switching_regime(x0, Alst, J)
    for i, state in enumerate(regime):
        state.append(i)

    names = ['x0', 'A0', 'A1', 'A2', 'xd']
    matrices = [x0, A0, A1, A2, xd]
    print(MP.String.multi_matrix(names, matrices))
    print('\nDeparture Sequence:')
    print(MP.String.regime_string(regime[20:]))

    print('\nResult:')

    print('\nRunning the 5-composite greedy method:')
    J_greedy = MP.Solve.p_greedy(Alst, regime, xd, 5, timer=True)

    print('\nRunning the combinatorial method:')
    J_comb = MP.Solve.combinatorial_method(Alst, regime, xd, timer=True, timeout=1)

    regime_greedy = MP.Regime.switching_regime(xd, Alst, J_greedy)
    for i, state in enumerate(regime_greedy):
        state.append(i)
    print(f'\nSolution with 5-greedy method: J = {J_greedy}')
    print(MP.String.regime_string(regime_greedy[20:]))


def example_20():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 4], [3, 3]]
    A2 = [[2, 5], [3, 2]]
    A3 = [[2, 4], [3, 2]]

    A = [A0, A1, A2, A3]
    B = [A0, A2]
    Omega = [A, B]
    H = [1, 0, 0, 1, 0, 0, 1, 0 ,0]

    x0 = [1, 0]
    xd = [3, 0]

    timetable = MP.Regime.regime(x0, A0, 10)
    res_seq = MP.Solve.multi_combinatorial(Omega, H, timetable, xd)

    print(f'R = {res_seq[0]}')


def example_22():
    A0 = [[2, 3], [5, eps]]
    A1 = [[2, 3], [4, eps]]

    B0 = [[2, 3], [6, eps]]
    B1 = [[2, 3], [5, eps]]

    C0 = [[3, 3], [5, eps]]
    C1 = [[3, 3], [4, eps]]
    A = [A0, A1]
    B = [B0, B1]
```

```python
    C = [C0, C1]

    x0 = [0, 1]
    H = [0,0,0,0,0]
    Ht = [1, 0, 0, 0, 0]
    Htt = [2, 0, 0, 0, 0]
    J = [0,0,0,0,0]

    print(MP.Solve.power(A0))

    omega = [A, B, C]
    regime = MP.Regime.multi_switching_regime(x0, omega, H, J)
    dregime = MP.Regime.multi_switching_regime(x0, omega, Ht, J)
    ddregime = MP.Regime.multi_switching_regime(x0, omega, Htt, J)

    print(MP.String.regime_string(regime))
    print()
    print(MP.String.regime_string(dregime))
    print()
    print(MP.String.regime_string(ddregime))


def example_25():
    A0 = [[2, 5], [3, 3]]
    A1 = [[2, 16], [3, 3]]
    A2 = [[2, eps], [3, 3]]

    x0 = [1,0]

    Omega = [[A0, A2], [A1, A2]]
    H = [0, 0, 0, 0, 0, 0]
    Hd = [1, 0, 0, 0, 0, 0]
    J = [0, 0, 0, 0, 0, 0]
    Jd = [1, 0, 0, 0, 0, 0]

    regime = MP.Regime.multi_switching_regime(x0, Omega, H, J)
    delayed_regime = MP.Regime.multi_switching_regime(x0, Omega, Hd, J)
    resolved_regime = MP.Regime.multi_switching_regime(x0, Omega, Hd, Jd)

    print('Regime:')
    print(MP.String.regime_string(regime))
    print('\nDelayed Regime:')
    print(MP.String.regime_string(delayed_regime))
    print('\n Resolved Regime:')
    print(MP.String.regime_string(resolved_regime))


def example_26():
    A_0 = [[2, 5], [3, 3]]
    A_d = [[2, eps], [3, 3]]
    x0 = [1, 0]
    Omega = [[A_0], [A_d]]
    H = [1, 0, 0, 0, 0]
    J = [0, 0, 0, 0, 0]

    print(MP.String.multi_matrix(['x0', 'A_0', 'A_d'], [x0, A_0, A_d]))

    print('\nResult: \n')

    regime = MP.Regime.multi_switching_regime(x0, Omega, H, J)

    print('Departure sequence:')
    print(MP.String.regime_string(regime))


def example_101():
    A0 = [[5, eps], [eps, 5]]
    A1 = [[3, eps], [eps, 3]]
    A2 = [[2, eps], [eps, 2]]
    Alst = [A0, A1, A2]
    Alst2 = [A2, A1, A0]
```

```
    Omega = [Alst, Alst2]

    x0 = [0, 0]
    xd = [100, 100]
    J = [0 for i in range(34)]
    H = [0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0]
    regime = MP.Regime.switching_regime(x0, Alst, J)
    for i, state in enumerate(regime):
        state.append(i)

    names = ['x0', 'A0', 'A1', 'A2', 'xd']
    matrices = [x0, A0, A1, A2, xd]
    print(MP.String.multi_matrix(names, matrices))
    print('\nDeparture Sequence:')
    print(MP.String.regime_string(regime[20:]))

    print('\nResult:')

    print('\nRunning the 5-composite greedy method:')
    J_greedy = MP.Solve.multi_p_greedy(Omega, H , regime, xd, 5, timer=True)
    print(J_greedy)

    print('\nRunning the 5-composite greedy method:')
    J_greedy = MP.Solve.p_greedy(Alst, regime, xd, 5, timer=True)
    print(J_greedy)

    print('\nRunning the combinatorial method:')
    J_comb = MP.Solve.combinatorial_method(Alst, regime, xd, timer=True, timeout=1)

    regime_greedy = MP.Regime.switching_regime(xd, Alst, J_greedy)
    for i, state in enumerate(regime_greedy):
        state.append(i)
    print(f'\nSolution with 5-greedy method: J = {J_greedy}')
    print(MP.String.regime_string(regime_greedy[10:]))
```

D.3. Simulation

This section contains the code for the simulation discussed in chapter 6. The code can be run through the `main.py` file.

D.3.1. Simulation Initialisation

```
from classes import MaxPlus, eps
from functions import get_arcs

A_0 = [[eps, eps, eps, eps, eps, 12 ],
           [10  , eps, 4  , eps, eps, eps],
           [eps, eps, 6  , 9  , eps, eps],
           [eps, eps, eps, eps, eps, 5  ],
           [eps, 4  , eps, eps, eps, eps],
           [eps, 8  , eps, eps, 4  , eps]]

def init_Omega():
    Alst0 = [A_0]
    arcs = get_arcs(A_0)

    for arc in arcs:
        i = arc[0]
        j = arc[1]
        A = MaxPlus.Operation.track_delay(A_0, (i,j), -1)
        Alst0.append(A)
        if A[i][j] >= 9:
            A = MaxPlus.Operation.track_delay(A_0, (i, j), -1)
            Alst0.append(A)

    Alst1 = [A_0]
    arcs = get_arcs(A_0)
    for arc in arcs:
        i = arc[0]
```

```
        j = arc[1]
        A = MaxPlus.Operation.track_delay(A_0, (i,j), -1)
        Alst1.append(A)

    Alst2 = [A_0]

    Omega = [Alst0, Alst1, Alst2]
    return Omega
```

D.3.2. Simulation

```
from classes import MaxPlus as MP, eps, Probability, Statistics
from functions import count
from simulation_data import init_Omega, A_0
import matplotlib.pyplot as plt
import scipy.interpolate
import random


class Simulation:
    def __init__(self):
        self.Omega = None
        self.J = None
        self.regime = None
        self.H = None
        self.stoch = None
        self.decouple = None

        self.init_omega()
        self.get_regime()
        self.get_H()
        self.init_prob()
        self.init_decouple()

    def init_omega(self):
        done = False
        while not done:
            done = True
            ret = input('Allow Switching? (Y/N) ')
            if ret == 'Y' or ret == 'y':
                self.Omega = init_Omega()
            elif ret == 'N' or ret == 'n':
                self.Omega = [[A_0]]
            else:
                print('Invalid input')
                done = False

    def get_regime(self):
        self.J = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        ev, self.x0 = MP.Solve.power(A_0)
        self.regime = MP.Regime.regime(self.x0, A_0, 10)

    def get_H(self):
        done = False
        while not done:
            done = True
            ret = input('Allow Shifting? (Y/N) ')
            if ret == 'Y' or ret == 'y':
                self.H = [1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0]
            elif ret == 'N' or ret == 'n':
                self.H = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
            else:
                print('Invalid input')
                done = False

    def init_prob(self):
        empty_p = {'TD': 0, 'SD': 0, 'TF': 0}
        onset_p = {'TD': 0.2, 'SD': 1/12, 'TF': 0.05}
        sustain_p = {'TD': 1/9, 'SD': 0, 'TF': 1/9}
        TD_distr = {1: 1, 2: 2, 3: 1}
```

```
        SD_distr = {1: 2, 2: 1}
        delay_distr = {'TD': TD_distr, 'SD': SD_distr}
        done = False
        while not done:
            done = True
            ret = input('Allow Delays? (Y/N) ')
            if ret == 'Y' or ret == 'y':
                self.stoch = [onset_p, sustain_p, delay_distr]
            elif ret == 'N' or ret == 'n':
                self.stoch = [empty_p, sustain_p, delay_distr]
            else:
                print('Invalid input')
                done = False

    def init_decouple(self):
        done = False
        while not done:
            done = True
            ret = input('Allow (Restricted) Decoupling? (R/Y/N) ')
            if ret == 'Y' or ret == 'y':
                self.decouple = [i for i in range(16)]
            elif ret == 'N' or ret == 'n':
                self.decouple = []
            elif ret == 'R' or ret == 'r':
                self.decouple = [0, 1, 5, 6, 7, 8, 9, 10, 14, 15]
            else:
                print('Invalid input')
                done = False

    def simulation(self, p, info=False, regime = False, seed=123, results=False):
        random.seed(seed)
        # print(f'Simulate: p = {p}')
        sim = MP.Simulation(self.x0, self.Omega, self.H, self.J, self.stoch, 16, 'P-Composite', p,
                            decouple=self.decouple, info=info)
        sim.simulate()

        DS = MP.Regime.delay_sequence(sim.regime, sim.equilibrium)
        delays = sim.delays

        if regime:
            print(MP.String.regime_string(sim.equilibrium, integer=True))
            print()
            print(MP.String.regime_string(sim.regime, integer=True))
            print()
            print(MP.String.regime_string(DS, integer=True))
            print()
            print(MP.String.regime_string(delays, integer=True))



        total_delay = []
        for state in DS:
            total_delay.append(sum(state))

        if results:
            print(f'Decoupled Trains: {sim.decoupled}')
            print(total_delay)
            print(f'Cumulative Total Delay: {sum(total_delay)}')
            print(f'Maximum Total Delay: {max(total_delay)}')
            print(f'SD Total Delay: {Statistics.SD(total_delay)}')

        return total_delay, sim.decoupled

    def simulation_various_p(self, rang, seed=123):
        total_delay_array = []
        decoupled_lst = []
        for p in range(rang):
            total_delay, decoupled = self.simulation(p, seed=seed)
            decoupled_lst.append(decoupled)
            total_delay_array.append(total_delay)
```

```
            print()

        print('p\t\tCTD\t\tMTD\t\tSDTD\t\t\t\t\tDT')
        for i, total_delay in enumerate(total_delay_array):
            print(f'{i}\t\t{int(sum(total_delay))}\t\t{int(max(total_delay))}\t\t{Statistics.SD(total_delay)}\t\t{decoupled_lst[i]}')


        p = 0
        plt.figure()
        for total_delay in total_delay_array:
            x = [i for i in range(len(total_delay))]
            xnew = [i/5 for i in range(len(total_delay)*5-4)]
            xx = [i/5+6 for i in range(len(total_delay)*5-4)]
            f = scipy.interpolate.interp1d(x, total_delay, kind='cubic')
            plt.plot(xx, f(xnew), label=p)
            plt.legend()
            p+=1
        plt.show()
        return total_delay_array, decoupled_lst

    def sample_simulation(self, p, time_check=False):
        TD = []
        CTD = []
        MTD = []
        SDTD = []
        DT = []
        for i in range(100):
            if time_check and i%10 == 0:
                print(f'{p}: Iteration {i}')

            total_delay, decoupled_lst = self.simulation(p, seed=i)
            TD.append(total_delay)
            CTD.append(sum(total_delay))
            MTD.append(max(total_delay))
            SDTD.append(Statistics.SD(total_delay))
            DT.append(decoupled_lst)
        return TD, CTD, MTD, SDTD, DT

# simulation_various_p()

    def stat_run_sim(self, rang):
        print('p\t\tCTD\t\t\tMTD\t\t\tSDTD\t\t\t\t\tDT')

        ylst = []

        for p in range(rang):
            TD, CTD, MTD, SDTD, DT = self.sample_simulation(p, time_check=True)
            print(p, end='\t\t')
            print(Statistics.mean(CTD), end='\t\t')
            print(Statistics.mean(MTD), end='\t\t')
            print(Statistics.mean(SDTD), end='\t\t')
            print(Statistics.mean(DT))

            y = []
            for i in range(len(TD[0])):
                horizontal = []
                for j in range(len(TD)):
                    horizontal.append(TD[j][i])
                y.append(Statistics.mean(horizontal))

            ylst.append(y)

        plt.figure()
        for p, y in enumerate(ylst):
            x = [i for i in range(len(y))]
            xnew = [i / 5 for i in range(len(y) * 5 - 4)]
            xx = [i / 5 + 6 for i in range(len(y) * 5 - 4)]
            f = scipy.interpolate.interp1d(x, y, kind='cubic')
            plt.plot(xx, f(xnew), label=f'p = {p}')
            plt.xlabel('Time (Hours)')
```

```
            plt.ylabel(’Time (6 minutes)’)
            plt.title(’The mean of \u0394(k) over 100 runs’)
            plt.legend()
        plt.show()

        plt.figure()
        for p, y in enumerate(ylst):
            x = [i for i in range(len(y))]
            plt.plot(x, y, label=f’p = {p}’)
            plt.xlabel(’Time (Hours)’)
            plt.ylabel(’Time (6 minutes)’)
            plt.title(’The mean of \u0394(k) over 100 runs’)
            plt.legend()
        plt.show()


# for i in range(len(reg)):
#     for j in range(len(reg[i])):
#         print(f’{int(reg[i][j]-delays[i][j])}’,end=’’)
#         if int(delays[i][j]) != 0:
#             print(f’\\r+{int(delays[i][j])}\\\\’)
#         else:
#             print(’\\\\’)
#     print()

# for i in range(len(DS)):
#     for j in range(len(DS[i])):
#         print(f’{int(DS[i][j])}\\\\’)
#     print()
```