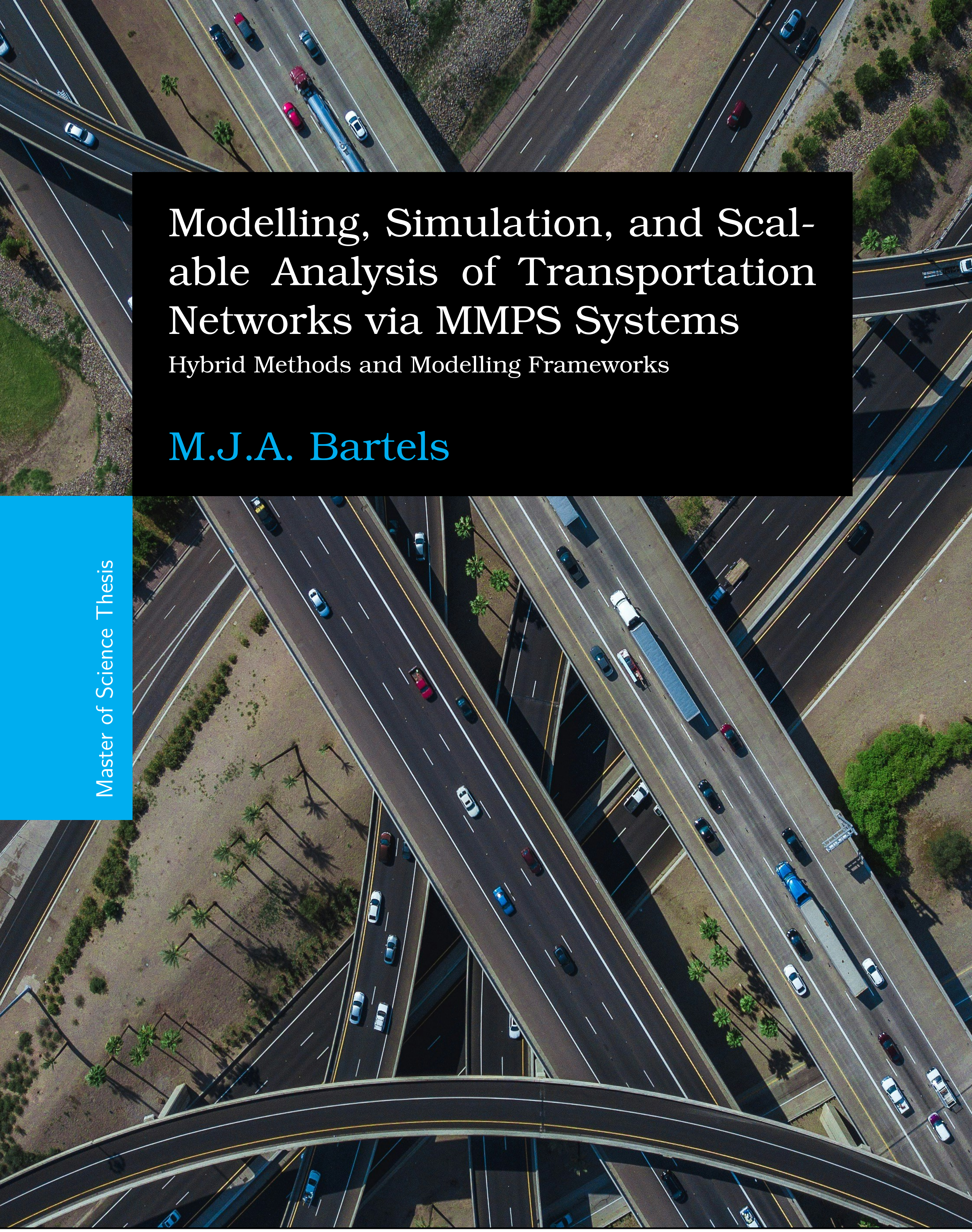
Modelling, Simulation, and Scalable Analysis of Transportation Networks via MMPS Systems

Hybrid Methods and Modelling Frameworks

M.J.A. Bartels

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control

Modelling, Simulation, and Scalable Analysis of Transportation Networks via MMPS Systems

Hybrid Methods and Modelling Frameworks

Master of Science Thesis

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

M.J.A. Bartels

September 25, 2025

Faculty of Mechanical Engineering (ME) · Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
DELFT CENTER FOR SYSTEMS AND CONTROL (DCSC)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical Engineering (ME) for acceptance a thesis entitled

MODELLING, SIMULATION, AND SCALABLE ANALYSIS OF TRANSPORTATION NETWORKS VIA MMPS SYSTEMS

by

M.J.A. BARTELS

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE SYSTEMS AND CONTROL

Dated: September 25, 2025

Supervisor(s):

dr.ir. A.J.J. van den Boom

ir. S. Markkassery

Reader(s):

dr. M. Guo

dr. B. Atasoy

Abstract

This thesis explores the analysis, periodicity, and scalable modelling of Max-Min-Plus-Scaling (MMPS) systems, a versatile approach to modelling Discrete Event (DE) systems. Unlike traditional continuous-time or discrete-time systems that evolve through differential or difference equations, DE systems progress through discrete events. MMPS systems rely only on maximisation, minimisation, addition, and scaling, making them highly suitable for modelling processes with synchronisation and/or competition such as energy delivery, transportation, and manufacturing.

The work is divided into three main segments. First, a new Mixed-Integer Linear Programming (MILP)-based method is developed for analysing growth rates and fixed points of general implicit MMPS systems. This extends an existing MILP formulation for homogeneous and non-expansive explicit MMPS systems, introducing adaptations for general implicit cases. A dedicated preprocessing step and search strategy are introduced, resulting in an analysis method that significantly reduces computational requirements. Secondly, the dynamical and stability behaviour of periodic MMPS systems with periods greater than one is examined. A new canonical form is proposed, enabling the use of existing analysis tools on periodic systems, along with a method for determining the stability of periodic orbits. Thirdly, a modelling framework for transportation systems is introduced, featuring a connectable, node-based toolbox and an algorithm that transforms high-level system descriptions into sets of equations.

All developed methods, theories, and tools are demonstrated on a real-world 4-node transportation system. The results confirm the efficiency of the new MILP approach, reveal periodic behaviour and stable periodic orbits, and highlight fixed points, all within the proposed transportation network framework.

The thesis follows a logical progression, starting with the mathematical preliminaries of MMPS systems, then presenting the main contributions, and concluding with key insights. Overall, it delivers new theoretical results for MMPS system analysis alongside practical tools and examples for modelling complex discrete-event systems, with a particular focus on transportation applications.

Table of Contents

Acknowledgements

Over the past year or so, I have been working on my thesis. It was something that I did not look forward to during my bachelor's and my master's. Focusing on just one subject and really diving deep into the matter seemed like an endless journey, for which I did not know if I had what it took to finish it. However, now that I have finished it, I feel a sense of pride. I have dedicated over 1500 hours to MMPS systems and have spent countless times explaining to friends and family what it is that I do. Where more time than not, they were probably left more confused than before. This thesis has been both a challenge and a passion, shaping not only my academic skills but also my time here in Delft. Completing my Master's feels like a bittersweet moment: as much as I look forward to finishing, it also marks the end of an unforgettable chapter in this city. A time which I have enjoyed deeply. And I don't think I would have been able to do it without all the support around me.

To start, I would like to thank my supervisors, Ton van den Boom and Sreeshma Markkassery, who have helped me throughout the entire process with their guidance and advice. Ton, the passion which which you speak about MMPS systems is extraordinary to see. It is this passion that has helped me see new opportunities to explore and give motivation whenever I didn't have it. To Sreeshma, you were always there to help and guide me during the entire process. Your encouragement and involvement have always been greatly appreciated. Every little or large bit of feedback was always given with the best intentions and without, I don't know how long it would have taken me on my own if ever. So Ton and Sreeshma, thank you.

To my girlfriend, Carine, who has supported me throughout my thesis with fun dates, dinners and lots of laughs. You were always able to cheer me up whenever my thesis was not going my way. But also helping me take breaks when things were going my way to keep a somewhat healthy work-life balance.

To my study friends, master friends, and 154 board members, Niels, Peter, Maartje, Essan, Sabine, Sep, Guus, Rutger, Nelis, Jessie, Els, Floris, Demi, Coen and especially Vicky, who is my fellow MMPS warrior. Without you, the countless days in landscape would have made me go crazy. I am very thankful I had all of you around me to take breaks with, sit in the sun, bitch about whatever wasn't working or just being. Without you, I don't think I would be able to call myself an engineer.

To my roommates, Floris, Ruben, Victor, Kasper and Johan, you have always made my house feel like a home. Even though with my busy schedule I was gone a lot, when I was home, I always enjoyed our time together to unwind. Were it drinking a beer on our roof terrace or playing games, I always had a blast, allowing me to start the next day fresh and relaxed.

To Heren 5, my hockey team, thank you for being there every Tuesday and Sunday, giving me the chance to blow off steam, stay fit, and escape my MMPS/ME bubble for a while. Not only keeping me moving, but also reminding me of the joy of good company, shared laughs, and the occasional questionable third half/ fifth quarter.

To my siblings, Olle and Imke, as siblings, we do not always get along, something that anyone with siblings will know. But thank you for always being there and having my back. Even though you have no clue what it is I do, but still support me regardless.

To my parents, without whom I would not have made it so far. Their undying support for what I do has always been a stable foundation on which I could build, and even though what I do sometimes sounds like magic to them, they understand the effort it takes to do it all, for which I am very thankful.

Thank you all.

Mees

Delft, University of Technology
September 25, 2025

“Failure is only the opportunity to begin again. Only this time, more wisely”
— *Uncle Iroh*

Chapter 1

Introduction

This Chapter provides an overview of what is discussed in this thesis. It begins with some relevant background information in Section 1-1, followed by the presentation of academic relevance and research questions in Section 1-2, where the research approach is also described. The chapter finishes with a document outline in Section 1-3.

1-1 Background

Every day, everybody comes into contact with control theory and control systems. All around us, we can find implementations of systems and control, sometimes more recognisable than others. Examples of control systems around us are energy delivery, transportation, manufacturing, medical devices and much more. In order to use and control these systems, a good model of the system is vital. Models allow us to capture the dynamics of a system, design a controller and control the real-world system to the desired output. Typically, these models are based on differential equations in conventional algebra and evolve over time due to the influence of various phenomena such as physical, chemical or biological phenomena.

However, a distinct subclass of systems, known as Discrete Event System (DES), evolve through discrete events rather than continuous time [1]. Examples include logistics networks, manufacturing lines, and urban railway systems [2]. Using conventional algebra to model such systems often leads to complex, non-linear descriptions that are difficult to work with. Max-plus and min-plus algebra can simplify this, as the synchronisation and competition effects that make DE systems highly non-linear become linear in these frameworks, greatly easing modelling and analysis.

When a DES is modelled purely as a max-plus or min-plus system, however, much of the modelling flexibility is lost. MMPS systems address this by combining max-plus, min-plus, and conventional algebra into a single powerful framework. This allows synchronisation, competition, and accumulation to be represented in the same system description, opening up a wide range of applications. MMPS systems capture both temporal states, such as the arrival time of vehicles and quantity states, such as the number of goods in that vehicle, while keeping a linear structure in their representation. They are mathematically equivalent to continuous piecewise-affine systems, making them a natural fit for modelling and controlling systems with hybrid dynamics.

1-2 Problem Description

The research into MMPS systems is currently in its infancy, meaning that there are a lot of unanswered questions regarding all domains of modelling, control, and stability. This means that there are also numerous opportunities to expand the current state of the art. In order to guide this research, some research questions are posed. Currently, the Urban Railway System (URS) is the only working example of MMPS systems in practice. This research will focus on extending this list of examples as well as diving into the analysis of MMPS systems. Since this becomes more complex, the larger the system gets. The concept of periodicity is also addressed, which leads to the following research questions.

1-2-1 Research Questions

1. How can a hybrid approach combining search trees, LPP, and MILP reduce the computational complexity of analysing Max-Min-Plus-Scaling Systems?

   (a) How can the existing explicit MILP algorithm be extended to also apply to general implicit MMPS systems?

   (b) How can a search tree be used to systematically explore and prune the search for eigenvalues of MMPS systems to avoid redundant or infeasible paths?

2. How can new theoretical insights into the structure and dynamics of MMPS systems contribute to more effective analysis of periodic system behaviour?

   (a) How can periodic MMPS systems be transformed to allow for periodicity and stability analysis?

   (b) How can the stability of periodic orbits be guaranteed?

3. Is it possible to model, simulate and analyse a transportation network such that it closely resembles reality?

   (a) Can the system equations be written in such a way as to incorporate all different arrival and departure patterns?

   (b) What insights can be obtained from analysing the dynamical behaviour of a 4-node transportation network?

   (c) How can the system be generalised to allow for more complex modelling, simulation and analysis?

   (d) Can a framework be created to allow for easy implementation of a generalised transportation network?

1-2-2 Approach

In Section 1-1 and 1-2, the background for this research is presented, and the research questions are posed. Before those questions can be addressed, the topic itself needs to be explored in more detail. This starts with a review of the current theory and literature, including an overview of the mathematical foundations of MMPS systems. Current analysis techniques are discussed, along with the known stability criteria for MMPS systems.

The next Chapter, Chapter 5, tackles research question 1. It begins by extending the existing MILP algorithm for topical MMPS systems to apply to general implicit MMPS systems, which answers research question 1(a). To make practical use of this MILP, a suitable search method is then developed, answering research question 1(b). Combining these elements results in a complete, stand-alone algorithm, which answers research question 1 as a whole.

The focus then shifts to the periodicity of MMPS systems. A general definition of periodic points, orbits, and behaviour is introduced, followed by a short discussion of the MMPS subclasses: max-plus and min-plus systems. This is extended to full MMPS systems, where existing analysis techniques currently fall short. A new canonical form is proposed that enables the use of all relevant methods, answering question 2(a). This also leads to new stability criteria for periodic orbits, addressing research question 2(b).

To include transport networks in the MMPS framework, a generalised implementation is developed that allows subsystems to be connected easily. The solvability and time invariance of these subsystems are investigated to ensure a working framework. Several example subsystems are provided to demonstrate practical use, answering research questions 3(c) and 3(d).

Finally, a detailed case study is carried out on a four-node transportation network to validate the earlier results. The system is modelled, addressing the challenge of describing all arrival and departure patterns and answering research question 3(a). An extensive analysis then follows, answering research question 3(b).

1-3 Document Outline

This thesis is organised to provide a clear and logical flow of reasoning, guiding the reader through the concepts in a natural way. The chapters are presented below in the order they appear, along with a short summary of their content. Chapters 2–4 give an overview of the existing literature, while Chapters 5–8 contain the main academic contributions of this work.

- **Chapter 2 - Preliminaries of MMPS Systems:** Introduces the algebraic frameworks max-plus and min-plus in both the scalar case as well as for matrices and vectors, which are a foundation of MMPS systems. MMPS functions and systems are introduced for the explicit and implicit case, but also for autonomous and controlled cases.
- **Chapter 3 - Analysis of MMPS Systems:** Contains key theory regarding system properties such as time-invariance, monotonicity and non-expansiveness is discussed. Differences between explicit and implicit systems are discussed, as well as the solvability conditions for implicit MMPS systems are given. Eigenvalue analysis is presented, and normalisation is introduced.

- **Chapter 4 - Stability of MMPS systems:** Presents existing theory on bounded-buffer stability, maximal invariant sets, and the conditions under which MMPS systems remain stable over time is reviewed. The discussion also covers linearisation, highlighting the differences between implicit and explicit system formulations.

- **Chapter 5 - Scalable Analysis of MMPS systems:** Gives an introduction to MILP and the differences with Linear Programming Problem (LPP) are discussed. The concept of MMPS systems, modes and dominant modes is introduced. A new algorithm for general time-invariant implicit MMPS systems is presented and derived. This chapter also introduces a search algorithm to effectively use the newly introduced MILP. The chapter finishes with some analysis on this new method as well as a run time comparison between the MILP and LPP methods for MMPS systems.

- **Chapter 6 - Periodicity of MMPS systems:** Extends the concept of periodicity to MMPS systems with a period of more than 1. Some remarks are made, and examples are given. The maximum period length for max-plus systems is extended to min-plus systems. The periodic behaviour of general MMPS systems is investigated. First, a new canonical form is introduced to allow for the application of current analysis methods. The concept of semi-dominant modes is introduced. Finally, the effects of periodic in MMPS systems on normalisation and linearisation are investigated, where a stability criterion for periodic MMPS systems is presented.

- **Chapter 7 - Modelling framework for Transportation Networks:** Presents the basics of modular sub-systems for transportation networks. Both time invariance and solvability are discussed. A method of transforming a switching MMPS system into a single MMPS system under certain conditions is presented. After this, a framework for the modelling of transportation systems is given. First, several nodes are presented, then an algorithm to turn a high-level system description into a proper system of equations is presented.

- **Chapter 8 - Case Study Transportation Systems:** Models a complex transportation system as well as performs an extensive analysis using all newly presented methods and techniques. The model is validated through simulation and demonstrates the real-world application potential.

- **Chapter 9 - Conclusions and Contributions:** Offers a brief reflection on each research question, summarising the corresponding answers and providing a concise overview of the work conducted throughout the thesis.

- **Chapter 10 - Recommendations for Future Work:** Discusses direction for future research

Chapter 2

Preliminaries of MMPS Systems

The goal of this chapter is to build a clear and complete mathematical foundation for understanding MMPS systems. Section 2-1 starts by introducing what a discrete event system is. Section 2-2 starts with the basics of max-plus algebra, covering what defines the algebra, how its operators behave, and its key properties, all introduced in the scalar case. Which then gets expanded to vectors and matrices.

Then the focus is shifted to min-plus algebra by analogy, which also gets expanded to vectors and matrices. The chapter concludes with an introduction to MMPS systems by breaking it up into its individual components, which then brings it together in a vector-valued form in Section 2-3.

2-1 Discrete Event Systems

DES form a broad class of dynamical systems characterised by their evolution being driven by discrete events rather than continuous flows or fixed time steps. These systems stand in contrast to the more familiar discrete-time systems [3].

In discrete-time systems, state changes occur at predetermined intervals based on a fixed sampling time that remains constant over a given time series. However, in discrete-event systems, state changes are triggered by specific events, which can happen at irregular intervals. As a result, the time between successive events is not fixed, and the system's dynamics are inherently event-driven. This means that the state itself has the dimension of time, and the steps equate to the counter for the events. Notice that this is flipped with respect to discrete-time systems.

Examples of DES include manufacturing systems, traffic networks, and communication protocols, where the timing and sequencing of events are crucial to their operation [2]. Understanding these systems is essential for modelling and analysing processes where discrete events dominate the dynamics.

2-2 Fundamentals of Dioids

Dioids provide a mathematical framework to model and analyse systems like discrete event systems. It incorporates operations that are tailored for certain applications, such as scheduling, optimisation, and control theory.

Semirings are an algebraic structure often used for modelling systems such as formal language, optimisation problems and hybrid systems.

Definition 2.1. *(Semiring [1])*
A semiring is a nonempty set R endowed with two binary operations $\oplus_R$ and $\otimes_R$ such that

- $\oplus_R$ *is associative and commutative with zero element* ε_R*;*
- $\otimes_R$ *is associative, distributes over* $\oplus_R$*, and has unit element* e_R*;*
- ε_R *is absorbing for* $\otimes_R$*.*

Such a semiring is denoted by $\mathcal{R} = (R, \oplus_R, \otimes_R, \varepsilon_R, e_R)$*.*

Associative means that it does not matter how elements are grouped. So the ordering is not important. Multiplication is associative since $(4 \times 3) \times 2 = 4 \times (3 \times 2)$. *Commutative* means that the order of the elements in the operation does not matter. For example, addition is commutative, since $5 + 3 = 8 = 3 + 5$.

Dioids are a special subclass of semiring. Dioids have as an extra condition that their addition is idempotent, which means that $a \oplus_R a = a$.

2-2-1 Max-Plus Algebra

Max-Plus algebra is a type of dioid that operates using maximisation and addition on the set of real numbers extended with $-\infty$ as such: $\mathbb{R}_\varepsilon = \mathbb{R} \cup \{-\infty\}$. The two operations are denoted by $\otimes$ ('*otimes*') and $\oplus$ ('*oplus*') respectively so

$$\begin{aligned} a \oplus b &= \max(a, b) \\ a \otimes b &= a + b \end{aligned} \tag{2-1}$$

where $a, b \in \mathbb{R}_\varepsilon$. $\varepsilon \overset{\text{def}}{=} -\infty$ and $e \overset{\text{def}}{=} 0$ are defined as the neutral elements with respect to $\otimes$ and $\oplus$ [1]. In some literature, $\mathbb{R}_\varepsilon$ is referred to as $\mathbb{R}_{\max}$; they are, however, the same.

These operations form the basis of max-plus algebra and exhibit properties analogous to those of conventional algebra, such as associativity and distributivity.

The set $\mathbb{R}_\varepsilon$ together with the operations $\oplus$ and $\otimes$ is called *max-plus algebra* and is denoted by

$$\mathcal{R}_{\max} = (\mathbb{R}_\varepsilon, \oplus, \otimes, \varepsilon, e) \tag{2-2}$$

Just as in conventional algebra, $\otimes$ has priority over $\oplus$.

Both operations are *commutative* since clearly

$$\begin{aligned} \max(a,b) &= \max(b,a) \\ a+b &= b+a \end{aligned} \tag{2-3}$$

For max-plus algebra, it is easy to see that there is no inverse element for $\oplus$. Take $a \oplus b = b$, if one is given b, one can never know what a was, only that it was less than b.

A definition for max-plus algebraic powers is introduced. For $a \in \mathbb{R}_\varepsilon$ and $n \in \mathbb{R}$, note that if n is an element of $\mathbb{R}$ and not $\mathbb{Z}^+$, so it is also possible to have negative powers or powers of non-natural numbers. Then the max-plus power is defined as [1];

$$a^{\otimes n} = \underbrace{a + a + \cdots + a}_{n \text{ times}} = n \times a \tag{2-4}$$

For $a^{\otimes 0}$, this will be defined as $e = 0$. Which makes sense if one looks at (2-4). Then also by definition $\varepsilon^{\otimes e} = e = 0$ and $\varepsilon^{\otimes r} = \varepsilon$ for $r > e$. Max-plus algebraic powers have priority over max-plus multiplication and max-plus addition, which is equivalent to the priorities of conventional algebraic operations.

2-2-2 Vectors and Matrices in Max-Plus Algebra

The current max-plus operations $\oplus$ and $\otimes$ can be extended to matrices and matrix operations[1]. Define a $n \times m$ matrix $A \in \mathbb{R}_\varepsilon^{n\times m}$, where $n, m \in \mathbb{Z}^+$ and define $\underline{n} \overset{\text{def}}{=} \{1, 2, \ldots, n\}$. Then the matrix A can be written as

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \tag{2-5}$$

Where the ij^{th} element of matrix A, a_{ij} will often be denoted by $[\mathrm{A}]_{ij}$, $\forall i \in \underline{n}, j \in \underline{m}$.

The three most frequently used matrix operations are summation, multiplication and matrix powers. These operations are defined as follows

Definition 2.2. *(Max-plus matrix summation [1])*
The sum of matrices $A, B \in \mathbb{R}_\varepsilon^{n\times m}$, denoted by $A \oplus B$, is defined as

$$[A \oplus B]_{ij} = [A]_{ij} \oplus [B]_{ij} = \max\left([A]_{ij}, [B]_{ij}\right) \tag{2-6}$$

Definition 2.3. *(Max-plus matrix multiplication [1])*
For matrices $A \in \mathbb{R}_\varepsilon^{n\times l}$ and $B \in \mathbb{R}_\varepsilon^{l\times m}$, the matrix product $A \otimes B$ is defined as

$$\begin{aligned} [A \otimes B]_{ik} &= \bigoplus_{j=1}^{l} a_{ij} \otimes b_{jk} \\ &= \max_{j \in \underline{l}} \left(a_{ij} + b_{jk}\right) \end{aligned} \tag{2-7}$$

Definition 2.4. *(Max-plus matrix power [1])*
For a matrix $A \in \mathbb{R}_\varepsilon^{n \times m}$ The k^{th} power of the matrix, $A^{\otimes k}$, is defined as

$$A^{\otimes k} \stackrel{def}{=} \underbrace{A \otimes A \otimes \cdots \otimes A}_{k \ times} \tag{2-8}$$

In Subsection 2-2-1, the neutral elements for max-plus scalar operations are defined. The same can be done for the corresponding matrix operations. Define an identity matrix E in max-plus algebra as well as a null matrix $\mathcal{E}$.

$$E = \begin{bmatrix} e & \varepsilon & \dots & \varepsilon \\ \varepsilon & e & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon & \varepsilon & \dots & e \end{bmatrix} \qquad \mathcal{E} = \begin{bmatrix} \varepsilon & \varepsilon & \dots & \varepsilon \\ \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon & \varepsilon & \dots & \varepsilon \end{bmatrix} \tag{2-9}$$

Where $E \in \mathbb{R}_\varepsilon^{n \times n}$ and $\mathcal{E} \in \mathbb{R}_\varepsilon^{n \times m}$.

It is also easy to see that these matrices for any $A \in \mathbb{R}_\varepsilon^{n \times m}$ satisfy

$$\begin{aligned} A \oplus \mathcal{E}(n, m) = A = \mathcal{E}(n, m) \oplus A \\ A \otimes E(m, m) = A = E(n, n) \otimes A \end{aligned} \tag{2-10}$$

Note that for $k \geq 1$ the following holds

$$A \otimes \mathcal{E}(m, k) = \mathcal{E}(n, k) \quad \text{and} \quad \mathcal{E}(k, n) \otimes A = \mathcal{E}(k, m) \tag{2-11}$$

Furthermore, combining Definition 2.4 and (2-9) we define the following

$$A_{n \times n}^{\otimes 0} \stackrel{\text{def}}{=} E_{n \times n} \tag{2-12}$$

Scalar–vector products are always interpreted element-wise. This applies to both the $\otimes$ and $\oplus$ operators, meaning that multiplying a scalar with a vector using either of these operators results in each element of the vector being scaled individually. Lastly, a matrix $A \in \mathbb{R}_\varepsilon^{n \times m}$ is called **regular** if A contains at least one element different from ε in each row.

2-2-3 Min-Plus Algebra

Another dioid is min-plus algebra. This algebra is very similar to the max-plus algebra, except for the $\oplus_\mathcal{R}$ operation and the zero element.

In max-plus algebra, $\varepsilon = -\infty$ is used as the zero element and extends the set of real numbers $\mathbb{R}$. In min-plus this is done, but with $+\infty$. Define $\top = \infty$ as the zero element and let $\mathbb{R}_\top$ be $\mathbb{R} \cup \{\top\}$. Then in min-plus algebra, the addition term $\oplus_R$ is denoted by $\oplus'$ ('*oplus prime*') performs the min operation. The times operation $\otimes_R$ is denoted by $\otimes'$ ('*otimes prime*') performs the plus operation;

$$\begin{aligned} a \oplus' b &= \min(a, b) \\ a \otimes' b &= a \otimes b = a + b \end{aligned} \tag{2-13}$$

The set $\mathbb{R}_\top$ together with the operations $\oplus'$ and $\otimes'$ is called *min-plus algebra* and is denoted by

$$\mathcal{R}_{\min} = (\mathbb{R}_\top, \oplus', \otimes', \top, e) \tag{2-14}$$

A few mathematical equivalences regarding max-plus and min-plus representations are given below, which can help simplify modelling and analysis;

- $-\min(a, b) = \max(-a, -b)$
- $-\max(a, b) = \min(-a, -b)$
- $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- $\max(c, \min(a, b)) = \min(\max(c, a), \max(c, b))$
- $\min(c, \max(a, b)) = \max(\min(c, a), \min(c, b))$

This new min-plus algebra can also be extended to matrices and vectors in a similar way as with max-plus algebra, except the maximisation is changed to a minimisation, resulting in the following definitions for min-plus matrix summations and additions;

Definition 2.5. *(Min-plus matrix summation and addition [1])*
The sum of matrices $A, B \in \mathbb{R}_\top^{n\times m}$, denoted by $A \oplus' B$, is defined as

$$[A \oplus' B]_{ij} = [A]_{ij} \oplus' [B]_{ij} = \min\left([A]_{ij}, [B]_{ij}\right) \tag{2-15}$$

The multiplication of matrices $A \in \mathbb{R}_\top^{m\times n}$, $B \in \mathbb{R}_\top^{n\times p}$ denoted by $A \otimes' B$, is defined as

$$\left[A \otimes' C\right]_{ij} = \min_k \left([A]_{ik} + [C]_{kj}\right) \tag{2-16}$$

Lastly, the min-plus identity (E') and zero ($\mathcal{E}'$) matrices are given by;

$$E' = \begin{bmatrix} e & \top & \dots & \top \\ \top & e & \dots & \top \\ \vdots & \vdots & \ddots & \vdots \\ \top & \top & \dots & e \end{bmatrix} \qquad \mathcal{E}' = \begin{bmatrix} \top & \top & \dots & \top \\ \top & \top & \dots & \top \\ \vdots & \vdots & \ddots & \vdots \\ \top & \top & \dots & \top \end{bmatrix} \tag{2-17}$$

Where $E' \in \mathbb{R}_\top^{n\times n}$ and $\mathcal{E}' \in \mathbb{R}_\top^{n\times m}$. It is also easy to see that these matrices for any $A \in \mathbb{R}_\top^{n\times m}$ satisfy

$$\begin{aligned} A \oplus \mathcal{E}'(n, m) = A = \mathcal{E}'(n, m) \oplus A \\ A \otimes E'(m, m) = A = E'(n, n) \otimes A \end{aligned} \tag{2-18}$$

Note that for $k \geq 1$ the following holds

$$A \otimes \mathcal{E}'(m, k) = \mathcal{E}'(n, k) \quad \text{and} \quad \mathcal{E}'(k, n) \otimes A = \mathcal{E}'(k, m) \tag{2-19}$$

2-3 MMPS Systems

An MMPS system is a DES. The state has the dimension of time, while the counter k is the event counter. This section introduces all the different operations that are possible in an MMPS system, then this is extended to MMPS function and eventually MMPS systems. As the name MMPS already suggests, all possible operations are maximisation, minimisation, addition and scaling, where each operation has its own purpose when modelling systems as an MMPS system.

Let us start by taking a close look at all the different operations which can appear in a MMPS function [3]. The arrow will represent an operation with processing time τ_i starting at $x_1(k)$ and finishing at $x_2(k)$, so both in the same event cycle k.

Maximization 1

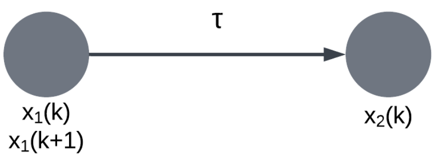


Figure 2-1: Maximisation 1 operator example

Consider an operation $x_1(k+1)$ that can not start until operation $x_1(k)$ has finished, as well as until the start signal $u_1(k+1)$ for operation $x_1(k+1)$ has been given. Thus, here the max operation comes into play. $x_1(k+1) = \max(x_1(k) + \tau, u_1(k+1))$

Maximization 2

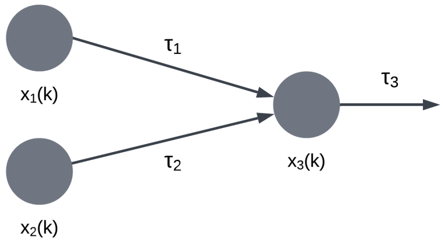


Figure 2-2: maximisation 2 operator example

If there is a third operation with starting time $x_3(k)$ and that can only start when operations one and two are finished, then this is also a max operation. $x_3(k) = \max(x_1(k)+\tau_1, x_2(k)+\tau_2)$

Minimization 1

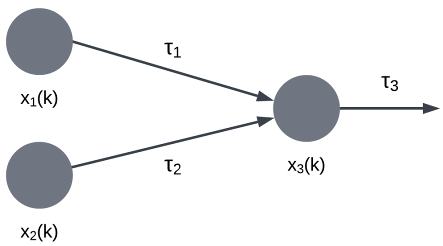


Figure 2-3: minimisation 2 operator example

If there is a third operation with starting time $x_3(k)$ and that can start when operation one of the previous operations has finished, then this is a min operation. $x_3(k) = \min(x_1(k) + \tau_1, x_2(k) + \tau_2)$

Addition 1

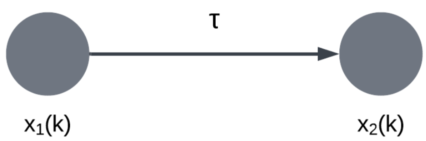


Figure 2-4: Addition operator example

Here, the relation between $x_1(k)$ and $x_2(k)$ is represented by the plus operation. $x_2(k)$ takes as long as $x_1(k)$ plus τ so $x_2(k) = x_1(k) + \tau$

Scaling 1

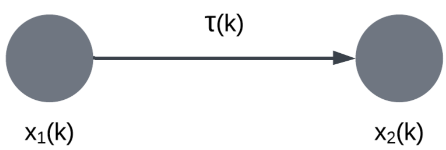


Figure 2-5: Scaling 1 operation example

When the operations processing time τ is an affine function of the state x, i.e $\tau(k) = \alpha + \beta^T x(k)$ where $\alpha \in \mathbb{R}_+$ and $\beta \in \mathbb{R}^n_+$ and where n is the dimension of the state. Then the relation between $x_1(k)$ and $x_1(k)$ includes a scaling operation. $x_2(k) = x_1(k) + \alpha + \beta^T x(k)$

Scaling 2

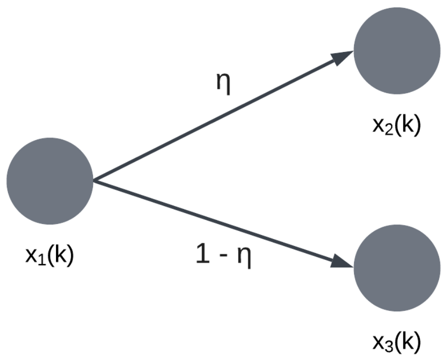


Figure 2-6: Scaling 2 operation example

When a state is split into two at a certain ratio. This is also a scaling operation. Let's say $x_1(k)$ gets split in $x_2(k)$ and $x_3(k)$ at a ratio of η and $1-\eta$ respectively. Then the scaling operation is give as $x_2(k) = \eta x_1(k), \quad x_3(k) = (1-\eta)x_1(k)$

To use these operations consistently, the set of real numbers is extended with $\top$ and ε by defining $\mathbb{R}_c = \mathbb{R} \cup \{-\infty\} \cup \{\infty\}$.

Together, these operations form the basis of MMPS functions. Which is defined as follows

Definition 2.6. *(Max-Min-Plus-Scaling function [3])*
A MMPS function is a function $f : \mathbb{R}_c^m \to \mathbb{R}_c$ is given by

$$f = x_i |\alpha| \max(f_k, f_l) |\min(f_k, f_l)| f_k + f_l \mid \beta \cdot f_k \tag{2-20}$$

with $i = 1, \ldots, m$, $\alpha, \beta \in \mathbb{R}$ and f_k and f_l both $\mathbb{R}_c^m \to \mathbb{R}_c$ MMPS functions. Where | stands for 'or'. Notice that this is a recursive definition. As one could have an infinite amount of nested MMPS functions.

Definition 2.7. *(Well defined MMPS function [3])*
An MMPS function $f : \mathbb{R}^m \to \mathbb{R}^n$ is well defined if the following holds:

$$\chi \in \mathbb{R}^m \quad \Longrightarrow \quad f(\chi) \in \mathbb{R}^n \tag{2-21}$$

Consider the following vector

$$\chi(k) = \left[x^T(k), x^T(k-1), \ldots, x^T(k-M), u^T(k), w^T(k)\right]^T \tag{2-22}$$

where $\chi \in \mathcal{X} \subseteq \mathbb{R}^{n_\chi}$, $x \in \mathbb{R}^n$ is the state vector. $u \in \mathbb{R}^p$ is the control input and $w \in \mathbb{R}^z$ is an external signal. The MMPS system is then described by

$$x(k) = f(\chi(k)) \tag{2-23}$$

Where f is a vector-valued MMPS function in variable χ and with event counter k.

Within the DES framework, MMPS systems have states that represent the starting and ending times of operations for event cycle k. However, in the general framework, the states may also include quantity states. For example number of people in a train or the goods in a production system. The basic operations will stay the same in this case. The state vector will just increase

$$x(k) = \begin{bmatrix} x_{\mathrm{t}}(k) \\ x_{\mathrm{q}}(k) \end{bmatrix} \tag{2-24}$$

Where $x_t(k)$ represents the time instances at which the k^{th} event occurs and $x_q(k)$ represents the values of the quantities at the k^{th} event occurrence.

As shown earlier, it is possible to describe an MMPS system in either implicit or explicit form. In the implicit case, the system includes a state vector such as in (2-22), where the state evolution depends on the current state. This is generally considered an undesirable property, as it complicates both computation and analysis. In many cases, an implicit MMPS system can be transformed into an explicit one through substitution. However, this transformation typically leads to an increase in system size and/or introduces a nested structure.

Chapter 3

Analysis of MMPS Systems

In the previous chapter, all the background for MMPS systems is given. This chapter focuses on the tools currently available for analysing the dynamical behaviour of MMPS systems. Section 3-1 discusses system properties such as time invariance, monotonicity and non-expansiveness, mainly for general vector-based systems. Section 3-2 dives into implicit MMPS systems; how they can be represented using a matrix-based description, which greatly simplifies analysis. It also discusses the system requirements for an implicit MMPS system to be solvable. Section 3-3 introduces the concept of additive eigenvalues for MMPS systems and how one can obtain them; first via the power algorithm, and then also via an LPP based approach for which normalisation is required and thus also introduced.

3-1 Explicit MMPS Systems

Explicit systems are systems where the state of the system only depends on the previous states and inputs. Explicit systems are a simplified version of implicit systems, which are easier to analyse. This section briefly introduces some properties of explicit MMPS system, such as time invariance and monotonicity.

$\mathbf{1}$ and $\mathbf{0}$ are used to denote a column vector with all elements equal to one and zero of the appropriate dimension, respectively. In some cases, the length of the vector is specified; this is done with a subscript. As an example, with $\mathbf{1}_n$, a vector of length n with all elements equal to one is meant.

The properties of time-invariance, monotonicity and non-expansiveness are important properties for the analysis of DES. These properties are defined as follows for general vector-based functions;

Definition 3.1. *(Homogeneity, monotonicity and non-expansive systems [4])*
Consider a system of the form $x(k+1) = f(x(k))$. This system is homogeneous if

$$f(x(k-1) + h\mathbf{1}) = f(x(k-1)) + h\mathbf{1} \qquad \forall h \in \mathbb{R} \tag{3-1}$$

f is monotonic if for any

$$x \leq y \quad \textit{then} \quad f(x) \leq f(y) \tag{3-2}$$

Lastly, f is non-expansive in the ℓ-norm if

$$\|f(x) - f(y)\|_\ell \leq \|x - y\|_\ell \tag{3-3}$$

For explicit MMPS systems, there are necessary conditions defined for when an MMPS system is time invariant, monotonic and/or non-expansive, which can be found in [4]. When an MMPS system is time invariant, monotonic and non-expansive, it is called topical.

3-2 Implicit MMPS Systems

The previous section provided a brief look at explicit MMPS systems and some of their basic properties. In contrast, the majority of MMPS systems encountered in practical applications are implicit. In these systems, the next state depends not only on previous states and inputs, but also on the current state itself. This self-dependence adds complexity to both the mathematical analysis and numerical processing of the system. The function-based description for MMPS systems introduced in Section 2-3 is hard to work with and error-prone; that is why an easy-to-work-with canonical form was developed. This form is called the ABCD canonical form, where the MMPS system has been transformed into a matrix-based state space description. This canonical form considers implicit MMPS systems of the form $x(k) = f(x(k), x(k-1))$.

Definition 3.2. *(ABCD-canonical form for implicit MMPS systems [5])*
The implicit ABCD canonical form describes an MMPS system as follows

$$x(k) = A \otimes (B \otimes' (C \cdot x(k-1) + D \cdot x(k))) \tag{3-4}$$

Where $x \in \mathbb{R}^n, A \in \mathbb{R}_\varepsilon^{n\times m}, B \in \mathbb{R}_\top^{m\times p}, C \in \mathbb{R}^{p\times n},\ D \in \mathbb{R}^{p\times n}$ and $k \in \mathbb{Z}^+$

Any implicit MMPS system can be written in this ABCD form [5]. Please note that this is not a unique description, as the ordering of the affine terms inside an MMPS function does not matter, so it is the same true for the ABCD canonical form. The presented ABCD form makes no distinction between the temporal states and the quantity states, but as is known, an MMPS system can have both, so we define a more specific form.

Definition 3.3. *(Implicit ABCD canonical form with temporal and quantity states [5])*
The implicit ABCD canonical form with temporal and quantity states MMPS system is described by

$$\begin{bmatrix} x_t(k) \\ x_q(k) \end{bmatrix} = \underbrace{\begin{bmatrix} A_t & \varepsilon \\ \varepsilon & A_q \end{bmatrix}}_{A} \otimes \Bigg(\underbrace{\begin{bmatrix} B_t & \top \\ \top & B_q \end{bmatrix}}_{B} \otimes' \Bigg(\underbrace{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{C} \cdot \begin{bmatrix} x_t(k-1) \\ x_q(k-1) \end{bmatrix} + \underbrace{\begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix}}_{D} \cdot \begin{bmatrix} x_t(k) \\ x_q(k) \end{bmatrix} \Bigg)\Bigg) \tag{3-5}$$

where $x_{\mathrm{t}} \in \mathbb{R}^{n_{\mathrm{t}}}, x_{\mathrm{q}} \in \mathbb{R}^{n_{\mathrm{q}}}, A_{\mathrm{t}} \in \mathbb{R}_{\varepsilon}^{n_{\mathrm{t}} \times m_{\mathrm{t}}}, A_{\mathrm{q}} \in \mathbb{R}_{\varepsilon}^{n_{\mathrm{q}} \times m_{\mathrm{q}}}, B_{\mathrm{t}} \in \mathbb{R}_{\top}^{m_{\mathrm{t}} \times p_{\mathrm{t}}}, B_{\mathrm{q}} \in \mathbb{R}_{\top}^{m_{\mathrm{q}} \times p_{\mathrm{q}}}, C_{11}, D_{11} \in \mathbb{R}^{p_{\mathrm{t}} \times n_{\mathrm{t}}}$, $C_{12}, D_{12} \in \mathbb{R}^{p_{\mathrm{t}} \times n_{\mathrm{q}}}, C_{21}, D_{21} \in \mathbb{R}^{p_{\mathrm{q}} \times n_{\mathrm{t}}}$, and $C_{22}, D_{22} \in \mathbb{R}^{p_{\mathrm{q}} \times n_{\mathrm{q}}}$. The notations ε and $\top$ represent matrices of appropriate sizes with all elements equal to ε and $\top$ respectively.

Alternatively, can one consider the different operations and elements as separate entities. This becomes useful when trying to change or identify certain properties. This is called the extended state MMPS system and is defined as follows;

Definition 3.4. *(Extended state MMPS system)*
An MMPS system can be represented in the following extended state form

$$\begin{aligned} x(k) &= A \otimes y(k) \\ y(k) &= B \otimes' z(k) \\ z(k) &= C \cdot x(k-1) + D \cdot x(k) \end{aligned} \tag{3-6}$$

3-2-1 Time Invariance of Implicit MMPS Systems

For Implicit MMPS systems to be time-invariant, they must also be additively partly homogeneous [5]. A property already introduced in Section 3-1. Following [5], explicit systems in the ABCD canonical form with temporal and quantity states are time-invariant if

$$\sum_{i \in \overline{n_{\mathrm{t}}}} [C_{11}\ D_{11}]_{\ell i} = 1, \forall \ell \in \overline{p_{\mathrm{t}}}, \quad \sum_{i \in \overline{n_{\mathrm{t}}}} [C_{21}\ D_{21}]_{ti} = 0, \forall t \in \overline{p_{\mathrm{q}}} \tag{3-7}$$

Where $[CD]$ represents the concatenation of the two matrices and not the matrix product. The proof can be found in [5].

For implicit MMPS systems, proofs and conditions for monotonicity and non-expansiveness do not exist yet.

3-2-2 Solvability of Implicit MMPS Systems

Implicit MMPS systems are difficult to work with. In general, however, writing the implicit system into an explicit one is possible. This will result in a nested MMPS system, which might be even more tedious to analyse, because the order of the system and complexity will increase quickly. However, not all implicit systems are solvable. An implicit MMPS system $x(k) = f(x(k-1), x(k))$ is solvable when [5]

$$x_i(k) = f_i\left(x(k-1), x_1(k), x_2(k), \ldots, x_{i-1}(k)\right) \forall i \in \bar{n} \tag{3-8}$$

Since a solution for $x_i(k)$ can be found by a finite successive substitution. In other words, the solution of $x_i(k)$ should not depend on $x_i(k)$. There is also another equivalent condition which uses structure matrices [5]. Which is as follows;

Take structure matrices S_A, S_B, S_D as;

$$\begin{aligned} &[S_A]_{i,j} = \begin{cases} 1 \text{ if } [A]_{i,j} \neq \varepsilon \\ 0 \text{ if } [A]_{i,j} = \varepsilon \end{cases} \qquad [S_B]_{i,j} = \begin{cases} 1 \text{ if } [B]_{i,j} \neq \top \\ 0 \text{ if } [B]_{i,j} = \top \end{cases} \\ &[S_D]_{i,j} = \begin{cases} 1 \text{ if } [D]_{i,j} \neq 0 \\ 0 \text{ if } [D]_{i,j} = 0 \end{cases} \end{aligned} \tag{3-9}$$

If there exists a matrix $T \in \mathbb{R}^{n\times n}$ such that $F = T \cdot S_A \cdot S_B \cdot S_D \cdot T^{-1}$, where F is a strict lower triangle matrix, there always exists a unique solution $x(k),\ k > 0$ for the implicit MMPS system for any state $x(k-1)$. Proof can be found in [5]. For any further analysis, it is assumed that the implicit MMPS system is solvable.

3-3 Eigenvalues and Eigenvectors

In conventional state space systems, analysis of their behaviour is often done using their eigenvalues and eigenvectors. Just like in conventional state-space systems, MMPS systems have eigenvalues that can be analysed to provide insights into their behaviour. Contrary to conventional multiplicative eigenvalues, MMPS systems are characterised by additive eigenvalues. Defined as follows;

Definition 3.5. *(Additive eigenvalues and eigenvectors [4])*
The time-invariant MMPS system $x(k) = f(x(k-1), x(k))$, $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^n$, with both temporal and quantity states, has an additive eigenvalue if there exists a real number $\lambda \in \mathbb{R}$ and a vector $v \in \mathbb{R}^n$ such that

$$f(v) = v + \lambda \cdot [\mathbf{1}_{n_t}^\top\ \mathbf{0}_{n_q}^\top]^\top \tag{3-10}$$

where n_c is the number of time states and n_q is the number of time states. Then the scalar λ is called the eigenvalue and the vector v is called the eigenvector.

Notice that since MMPS systems have additive eigenvalues and vectors, if v is an eigenvector then so is $v + h \cdot [\mathbf{1}_{n_t}^\top\ \mathbf{0}_{n_q}^\top]^\top \quad \forall h \in \mathbb{R}$. The eigenvalue of an MMPS system is also often referred to as the growth rate of the system. Whereas the eigenvector is often referred to as the fixed point. An eigenvalue whose existence solely depends on the structure of the matrices A, B, C and D is called a structural eigenvalue. This means that any numerical changes to the matrices will not change the existence of the additive eigenvalue. For topical MMPS systems, it is also known that there can only ever be a single additive eigenvalue [4]. The next two sections investigate two methods of obtaining the growth rate and fixed points of implicit MMPS systems. The first one is the Power Algorithm. The second method relies on normalising the MMPS system and solving a set of LPP problems.

3-3-1 Power Algorithm for Implicit MMPS Systems

The power algorithm relies on the system eventually ending in a periodic regime during simulation. It checks the difference between two state vectors, and once all time states have grown with the same amount and all quantity states remain the same over event cycles, a stationary regime has been found, and the algorithm terminates. The algorithm can be found in Algorithm 1.

One important note about the power algorithm is that when applied to an implicit MMPS system, the state can move to the growth rate in an asymptotic manner, which can lead to a never-stopping algorithm. That is why a maximum number of iterations can be beneficial.

Algorithm 1 Power Algorithm [4]

1. Take an arbitrary initial vector $x(0) = x_0 \neq \varepsilon \cdot \mathbf{1}$; that is, x_0 has at least one finite element.
2. Iterate $x(k) = f(x(k-1))$ until there are integers p, q with $p > q \geq 0$ and a real number c, such that $x_t(p) = x_t(q) \otimes c$ and $x_q(p) = x_q(q)$, i.e., until a periodic regime is reached.
3. Compute as the eigenvalue $\lambda = c/(p-q)$ (division in conventional sense).
4. Compute as an eigenvector $v = \bigoplus_{j=1}^{p-q} \left(\lambda^{\otimes(p-q-j)} \otimes x(q+j-1) \right)$

3-3-2 LPP Algorithm for Implicit MMPS Systems

Before the second algorithm is introduced, normalising an MMPS system must be introduced. After which, an LPP problem is presented to find the eigenvalue. Normalisation in an MMPS sense means that the system is transformed to have an eigenvalue of zero and an eigenvector at $\mathbf{0}$, thus when initialising the normalised system at the eigenvector $\mathbf{0}$, the system stays there. For this, it is important to know what a diagonal matrix looks like in an MMPS sense. Thus, they are defined below;

Definition 3.6. *(Max-plus and min-plus diagonal matrices [4]) Given a vector $v \in \mathbb{R}^n$ we define the max-plus diagonal matrix $d_\otimes(v)$ and the min-plus diagonal matrix $d_{\otimes'}(v)$ as*

$$d_\otimes(v) = \begin{bmatrix} v_1 & \varepsilon & \cdots & \varepsilon \\ \varepsilon & v_2 & & \vdots \\ \vdots & & \ddots & \vdots \\ \varepsilon & \cdots & \cdots & v_n \end{bmatrix}, d_{\otimes'}(v) = \begin{bmatrix} v_1 & \top & \cdots & \top \\ \top & v_2 & & \vdots \\ \vdots & & \ddots & \vdots \\ \top & \cdots & \cdots & v_n \end{bmatrix} \tag{3-11}$$

The inverse of the max-plus diagonal matrix is $d_\otimes(-v)$ and for min-plus $d_{\otimes'}(-v)$. Then the max-plus identity matrix E is also given by $d_\otimes(\mathbf{0})$. For the min-plus identity matrix one gets $d_{\otimes'}(\mathbf{0})$

Also, define the row-major order of a matrix. This means the mapping of a matrix to a column vector as such $\text{vec}(A) = [A_1^T A_2^T \dots A_n^T]^T$ with $A \in \mathbb{R}_c^{n\times m}$ where $A_i, i \in \overline{n}$ denotes the i^{th} row of matrix A.

Take a system of the form (3.3), if this system has an additive eigenvalue λ and additive eigenvector $(x_{te}, x_{qe}, y_{te}, y_{qe}, z_{te}, z_{qe})$. Then the following must hold;

$$\begin{aligned} z_{te} &= C_{11} \cdot (x_{te} - \lambda\mathbf{1}) + C_{12} \cdot x_{qe} + D_{11} \cdot x_{te} + D_{12} \cdot x_{qe} \\ z_{qe} &= C_{21} \cdot (x_{te} - \lambda\mathbf{1}) + C_{22} \cdot x_{qe} + D_{21} \cdot x_{te} + D_{22} \cdot x_{qe} \\ y_{te} &= B_t \otimes' z_{te}, \quad y_{qe} = B_q \otimes' z_{qe} \\ x_{te} &= A_t \otimes y_{te}, \quad x_{qe} = A_q \otimes y_{qe} \end{aligned} \tag{3-12}$$

Let $x_\mathrm{e} = \begin{bmatrix} x_\mathrm{te}^\top & x_\mathrm{qe}^\top \end{bmatrix}^\top, y_\mathrm{e} = \begin{bmatrix} y_\mathrm{te}^\top & y_\mathrm{qe}^\top \end{bmatrix}^\top, z_\mathrm{e} = \begin{bmatrix} z_\mathrm{te}^\top & z_\mathrm{qe}^\top \end{bmatrix}^\top$ and $s = \begin{bmatrix} \mathbf{1}_{n_\mathrm{t}}^\top & \mathbf{0}_{n_\mathrm{q}}^\top \end{bmatrix}^\top$. Define $x_{\mathrm{e},\lambda} = x_\mathrm{e} - s\lambda$ and $A_\lambda = \begin{bmatrix} A_{\mathrm{t},\lambda} & \varepsilon \\ \varepsilon & A_\mathrm{q} \end{bmatrix}$, $A_{\mathrm{t},\lambda} = A_{i_\mathrm{t} j_\mathrm{t}} - \lambda, \forall i_\mathrm{t} \in \overline{n_\mathrm{t}}, \forall j_\mathrm{t} \in \overline{m_\mathrm{t}}$.

Then (3-12) can be transformed to

$$\begin{aligned} w_\mathrm{e} &= (C + D) \cdot x_{\mathrm{e},\lambda} \\ y_\mathrm{e} &= B \otimes' (w_\mathrm{e} + D \cdot s\lambda) \\ x_{\mathrm{e},\lambda} &= A_\lambda \otimes y_e \end{aligned} \tag{3-13}$$

where $w_e = z_e - D \cdot s\lambda$. $y_\mathrm{e} = B \otimes' (w_\mathrm{e} + D \cdot s\lambda)$ can then be written as

$$y_\mathrm{e} = B' \otimes' w_\mathrm{e} \tag{3-14}$$

With $d = D \cdot s$ and $B' = B + \lambda \mathbf{1}_m \cdot d^\top$. Next, some diagonal matrices must be defined as such;

$$\begin{aligned} X &= \mathrm{d}_\otimes (x_{e,\lambda}), & X^{-1} &= \mathrm{d}_\otimes (-x_{\mathrm{e},\lambda}) \\ Y &= \mathrm{d}_\otimes (y_e), & Y^{-1} &= \mathrm{d}_\otimes (-y_e) \\ Y' &= \mathrm{d}_{\otimes'} (y_e), & (Y')^{-1} &= \mathrm{d}_{\otimes'} (-y_e) \\ W' &= \mathrm{d}_{\otimes'} (w_e), & (W')^{-1} &= \mathrm{d}_{\otimes'} (-w_e) \end{aligned} \tag{3-15}$$

Then it follows that the following is true;

$$\begin{array}{cc} X^{-1} \otimes x_{e,\lambda} = \mathbf{0} & Y^{-1} \otimes y_e = \mathbf{0} \\ (Y')^{-1} \otimes' y_e = \mathbf{0} & (W')^{-1} \otimes' w_e = \mathbf{0} \end{array} \tag{3-16}$$

By combining (3-13) and (3-15), one ends up with

$$\begin{aligned} X^{-1} \otimes x_{e,\lambda} &= X^{-1} \otimes A_\lambda \otimes y_e \\ &= \underbrace{X^{-1} \otimes A_\lambda \otimes Y}_{\tilde{A}} \otimes Y^{-1} \otimes y_e \\ Y^{-1} \otimes' y_e &= (Y')^{-1} \otimes' B \otimes' z_e \\ &= \underbrace{Y^{-1} \otimes' B' \otimes' W}_{\tilde{B}} \otimes' (W')^{-1} \otimes' w_e \end{aligned} \tag{3-17}$$

Due to the (3-16), this reduces to

$$\mathbf{0} = \tilde{B} \otimes' \mathbf{0}, \quad \mathbf{0} = \tilde{A} \otimes \mathbf{0} \tag{3-18}$$

And so the normalized MMPS system is found [4];

$$\tilde{z}(k) = (C \cdot \tilde{x}(k-1) + D \cdot \tilde{x}(k)), \quad \tilde{y}(k) = \tilde{B} \otimes' \tilde{z}(k), \quad \tilde{x}(k) = \tilde{A} \otimes \tilde{y}(k) \tag{3-19}$$

This system has an eigenvalue $\tilde{\lambda} = 0$ and an eigenvector of $\tilde{v}_e = \left(\tilde{x}_e^\top, \tilde{y}_e^\top, \tilde{z}_e^\top\right)^\top = \left(\mathbf{0}^\top, \mathbf{0}^\top, \mathbf{0}^\top\right)^\top$. This new normalised system will stay at zero when initiated at the eigenvector, which is also zero. Furthermore

$$\begin{aligned} x(k) &= \tilde{x}(k) + (k\lambda)s + x_\mathrm{e} \\ y(k) &= \tilde{y}(k) + (k\lambda)s + y_\mathrm{e} \\ z(k) &= \tilde{z}(k) + (k\lambda)s + z_\mathrm{e} \end{aligned} \tag{3-20}$$

It also follows from (3-18) that each row of $\tilde{A}$ has at least one zero on each row, as well as having all the other non-zero entries being smaller than zero. $\tilde{B}$ also has at least one element equal to zero in each row. Every other non-zero entry is larger than zero.

$$\min_l [\tilde{B}]_{jl} = 0 \quad \forall j, \quad \max_j [\tilde{A}]_{ij} = 0 \quad \forall i \tag{3-21}$$

Now that the system can be normalised, this property can be used in the development of the LPP algorithm. An MMPS system can have multiple additive eigenvalues and eigenvectors. This is similar to how non-linear systems in standard algebra can have several equilibrium points.

Let S be the number of additive eigenvalues. For each $\theta \in \{1, \ldots, S\}$, let λ_θ be the additive eigenvalue, and $\tilde{A}_\theta, \tilde{B}_\theta$ the corresponding normalized matrices.

To describe the structure of these configurations, footprint matrices are introduced. A footprint matrix shows the pattern of zeros in a normalised matrix and acts as a kind of blueprint. In the LPP algorithm, these matrices are used to guide the search for the growth rate and fixed point for a given system setup.

The footprint matrices are defined as follows;

$$\begin{aligned} &G_{A_\theta} = \begin{bmatrix} G_{A_{t\theta}} & \mathbf{0} \\ \mathbf{0} & G_{A_{q\theta}\theta} \end{bmatrix}, \quad G_{B_\theta} = \begin{bmatrix} G_{B_{t\theta}} & \mathbf{0} \\ \mathbf{0} & G_{B_{q\theta}\theta} \end{bmatrix} \\ &[G_{A_{t\theta}}]_{ij} = \begin{cases} 1 \text{ if } \left[\tilde{A}_{t\theta}\right]_{ij} = 0 \\ 0 \text{ if } \left[\tilde{A}_{t\theta}\right]_{ij} < 0 \end{cases}, [G_{B_{t\theta}}]_{jl} = \begin{cases} 1 \text{ if } \left[\tilde{B}_{t\theta}\right]_{jl} = 0 \\ 0 \text{ if } \left[\tilde{B}_{t\theta}\right]_{jl} > 0 \end{cases} \\ &\left[G_{A_{\mathrm{q}\theta}}\right]_{rs} = \begin{cases} 1 \text{ if } \left[\tilde{A}_{\mathrm{q}\theta}\right]_{rs} = 0 \\ 0 \text{ if } \left[\tilde{A}_{\mathrm{q}\theta}\right]_{rs} < 0 \end{cases}, \left[G_{B_{q\theta}}\right]_{st} = \begin{cases} 1 \text{ if } \left[\tilde{B}_{\mathrm{q}\theta}\right]_{st} = 0 \\ 0 \text{ if } \left[\tilde{B}_{\mathrm{q}\theta}\right]_{st} > 0 \end{cases} \end{aligned} \tag{3-22}$$

with $\mathbf{0}$ being a zero matrix of appropriate size. So a footprint matrix pair essentially refers to a system configuration. These footprint matrices are used by the LPP to find the eigenvalues. This is needed since the system configuration, which results in a valid growth rate and fixed point, is not known a priori, thus all configurations must be checked one by one by the LPP in the form of a recursive LPP implementation. Since there are $m_\mathrm{t}^{n_\mathrm{t}} p_\mathrm{t}^{m_\mathrm{t}} m_\mathrm{q}^{n_\mathrm{q}} p_\mathrm{q}^{m}$ number of footprint matrices, this is also the number of LPPs which need to be solved. Since any element in A equal to ε and any element in B equal to $\top$ act as the zero element, it means that those elements can never be equal to zero in a normalised system. This means that any footprint matrix where an element corresponding to a ε or $\top$ is equal to 1 is never valid and thus they can be removed from the list of possible footprint matrix, drastically reducing the number of possible footprint matrix pairs to;

$$\prod_{i=1}^{n} a_i \cdot \prod_{j=1}^{m} b_j \tag{3-23}$$

Where a_i denotes the number of finite elements in row i of matrix A and b_j denotes the number of finite elements in row j of matrix B. The LPP to find the eigenvalues and eigenvectors is given by [5];

$$\begin{aligned}
\min_{x_{\mathrm{e}}, y_e, w_e} \ & \lambda \\
\text{s.t.} \quad & -[s\lambda]_i - [x]_i + [y]_j \leq -[A]_{ij} \quad \text{if } [G_{\mathrm{A}_\theta}]_{ij} = 0 \\
& -[s\lambda]_i - [x]_i + [y]_j = -[A]_{ij} \quad \text{if } [G_{\mathrm{A}_\theta}]_{ij} = 1 \\
& [y]_j - [d]_\ell \lambda - [w]_\ell \leq [B]_{j\ell} \quad \text{if } [G_{\mathrm{B}_\theta}]_{j\ell} = 0 \\
& [y]_j - [d]_\ell \lambda - [w]_\ell = [B]_{j\ell} \quad \text{if } [G_{\mathrm{B}_\theta}]_{j\ell} = 1 \\
& d = D \cdot s, \quad w = (C + D) \cdot x
\end{aligned} \tag{3-24}$$

Where $s = \begin{bmatrix} \mathbf{1}_{n_{\mathrm{t}}}^\top & \mathbf{0}_{n_{\mathrm{q}}}^\top \end{bmatrix}^\top$. Vector s acts as a state classification vector denoting the difference between a temporal state and a quantity state. As it turns out, for a found valid footprint matrix pair, the solution to the LPP given by (λ^*, v^*) might not be the only one that satisfies the constraints of (3-24). Take the set of equalities and inequalities from the LPP and substitute the value for λ^*. One will end up with

$$H_{\mathrm{eq}} \cdot v = h_{\mathrm{eq}}, \quad H_{\mathrm{ineq}} \cdot v \leq h_{\mathrm{ineq}} \tag{3-25}$$

Matrix H_{eq} is a square matrix with a rank of at least one less than $m+n+p$, since the system is shift-invariant in the direction of $s = \begin{bmatrix} \mathbf{1}_{n_{\mathrm{t}}}^\top & \mathbf{0}_{n_{\mathrm{q}}}^\top \end{bmatrix}^\top$. When the rank is even lower, there are more directions in which the system is shift-invariant. This results in a set of fixed points for a given eigenvalue λ^*. Where the direction vectors are given by $g_1, g_2, \ldots, g_f$ with

$$g_1 = \begin{bmatrix} s \\ B \otimes' ((C + D) \cdot s) \\ (C + D) \cdot s \end{bmatrix} \tag{3-26}$$

This looks similar to s. However, it incorporates the extended states y_e and w_e as well. The fixed-point set $\mathcal{V}$ is given by

$$\mathcal{V} = \{ v \mid H_{\mathrm{ineq}} \cdot v \leq h_{\mathrm{ineq}} \} \tag{3-27}$$

With $v = v^* + \sigma_1 g_1 + \sigma_2 g_2 + \ldots \sigma_f g_f$ and $\sigma_1 \in \mathbb{R}, \ \sigma_i > 1 \quad \forall i \neq 1$. σ_i can be constrained to a specific range. Finding this range can be done using the inequality constraints. So in this subspace, the system is shift-invariant for that specific growth rate [5]. When one is only considering explicit MMPS systems, both the Power algorithm and the LPP algorithm are still valid to use. However, no one simply sets $D = \mathbf{0}$, which must have the same size as C.

Chapter 4

Stability of Max-Min-Plus-Scaling Systems

The stability of any system is vital for understanding the system's behaviour. Knowing whether states will converge or diverge is critical for making any decisions about systems. This chapter deals with the stability of MMPS systems, both implicit and explicit. Section 4-1 focuses on the steady state behaviour of MMPS systems, differentiating between how temporal state and quantity state behave under steady state. Then Section 4-2 introduces how to linearise an explicit MMPS system as well as determine the valid linearization region and presents existing theory regarding the stability criteria for explicit MMPS systems. Section 4-3 does the same but for implicit MMPS systems. Finishing the chapter, Section 4-4 describes a method how the maximal invariant set of a linearised MMPS system can be determined.

A DE system is stable when all the time states of the system grow at the same rate. This also means that the system is bounded-buffer stable [6]. Bounded buffer stability means that the system's buffer (or state values) remains within a finite range over time, ensuring that it does not grow uncontrollably regardless of inputs or disturbances.

To quantify this bound, the Hilbert projective norm is used. The Hilbert projective norm measures how much two positive vectors differ in their relative proportions, ignoring their overall scale, which is defined as follows:

Definition 4.1. *(Hilbert projective norm [1])*
The Hilbert projective norm of a vector $x \in \mathbb{R}^n$ is defined as

$$\|x\|_{\mathbb{P}} = \max_{i \in \underline{n}} (x_i) - \min_{j \in \underline{n}} (x_j) \tag{4-1}$$

This norm looks at the maximum difference between vectors. Therefore, evaluating whether the states do or do not diverge.

An autonomous DE system is bounded buffer stable if for every initial time state $x_{t0} \in \mathbb{R}^{n_t}$ there exists a bound $M(x_0) \in \mathbb{R}$ such that the temporal state is bounded in the Hilbert projective norm. i.e. $\|x_t(k)\|_{\mathbb{P}} \leq M(x_0) \quad \forall k \in \mathbb{Z}^+$.

4-1 Steady State Behaviour of Explicit MMPS Systems

Take a time-invariant explicit MMPS system, with both temporal and quantity states.

$$x(k) = \begin{bmatrix} x_{\mathrm{t}}(k) \\ x_{\mathrm{q}}(k) \end{bmatrix} = \begin{bmatrix} f_t\left(\chi_t(k), \chi_q(k)\right) \\ f_q\left(\chi_t(k), \chi_q(k)\right) \end{bmatrix} \tag{4-2}$$

Due to the different nature of the two states, temporal and quantitative, they will also have different stationary behaviour. A time signal is usually non-decreasing; thus, in general, a time signal will not reach an equilibrium. Instead for time signals are considered to have a stationary regime as a steady state [3]. That is, the growth of x_t becomes constant. That is for x_t and χ_t a stationary regime is reached if for a certain event k_{ss} the growth of x_t and χ_t becomes constant as:

$$\chi_{\mathrm{t}}(k) = \chi_{\mathrm{t}}(k-1) + \tau_{\mathrm{t},ss}\mathbf{1}, \text{ for } k \geq k_{\mathrm{ss}} \tag{4-3}$$

Where $\tau_{t,ss} \in \mathbb{R}$, and $\chi_t \in \mathbb{R}^m$

For the quantity states, the behaviour is different; instead of continuously growing, the state should eventually stabilise and become constant, as expected in normal steady-state conditions. This results in:

$$\chi_{\mathrm{q}}(k) = \chi_{\mathrm{q}}(k-1), \text{ for } k \geq k_{\mathrm{ss}} \tag{4-4}$$

Combining this one ends up with the following stationarity condition:

$$\begin{bmatrix} \chi_{\mathrm{t}}(k) \\ \chi_{\mathrm{q}}(k) \end{bmatrix} = \begin{bmatrix} \chi_{\mathrm{ss,t}} + k\tau_{\mathrm{ss,t}}\mathbf{1} \\ \chi_{\mathrm{ss,q}} \end{bmatrix}, \text{ for } k \geq k_{\mathrm{ss}} \tag{4-5}$$

Which will give the steady state values $x_{\mathrm{ss,t}}, x_{\mathrm{ss,q}}, \chi_{\mathrm{ss,t}}, \chi_{\mathrm{ss,q}}, \tau_{ss,t}$ [3]

This is similar to fixed points and growth rates in autonomous systems. However, unlike autonomous systems that operate without external influence, this scenario also accounts for inputs and disturbances. This can easily be seen by the fact that the system works with $\chi(k)$ instead of $x(k)$ in the MMPS functions f_t and f_q. Notice that if the system in (4-2) only depends on the previous and/or current state, i.e. $\chi(k) = [x(k-1), x(k)]$, then the stationarity conditions as the same as for eigenvalues and eigenvectors.

4-2 Bounded Buffer Stability of Explicit MMPS Systems

Chapter 3 shows how to normalize an MMPS system with respect to different growth rates. This normalised system can be transformed into a linearised system in conventional algebra. Which then can be used to determine the stability of an MMPS system around a specific fixed point. However, this linearised system is only valid within a to be specified polyhedron Ω_θ. Take an normalized implicit MMPS system:

$$\tilde{x}_\theta(k) = \tilde{A}_\theta \otimes \left(\tilde{B}_\theta \otimes' \left(C \cdot \tilde{x}_\theta(k-1)\right)\right) \tag{4-6}$$

For $\theta \in \{1, \ldots, S\}$, and where $\tilde{A}_\theta \in \mathbb{R}^{n\times m}$, $\tilde{B}_\theta \in \mathbb{R}^{m\times p}$, $C \in \mathbb{R}^{p\times n}$. Then this explicit MMPS system can be linearised as such:

Definition 4.2. *(Linearization of an explicit MMPS system [6])*
Any normalised explicit MMPS system can be linearised as follows:

$$\tilde{x}_\theta(k) = M_\theta \cdot \tilde{x}_\theta(k-1), \quad M_\theta = G_{A_\theta} \cdot G_{B_\theta} \cdot C \tag{4-7}$$

for all $\tilde{x}(k) \in \Omega_\theta,\ k \in \mathbb{Z}^+$

Notice that G_{A_θ} and G_{B_θ} are the footprint matrices with exactly one entry equal to one in each row. It is only possible to have multiple entries equal to one when the equilibrium point is on the boundary of two or more regions.

The linearization is only valid when $\tilde{x}(k) \in \Omega_\theta,\ k \in \mathbb{Z}^+$. Ω_θ can be described by a set of linear inequalities. However, before that is possible, the Kronecker vector product should be introduced. This will be defined below;

Definition 4.3. *(Vector Kronecker product [5])*
The Kronecker product of a matrix $A \in \mathbb{R}^{n\times m}$ *and a vector* $\mathbf{1}_p$, $A \boxtimes \mathbf{1}_p$, *stacks* p *copies of every row of the matrix* A *vertically.* $\mathbf{1}_p \boxtimes A$ *stacks* p *copies of the entire matrix vertically.*

Now that the Kronecker product has been introduced, the structure of the inequalities that define the polyhedral region Ω_θ can be examined. Ω_θ is described by:

$$\Omega_\theta = \{\tilde{x} | H \cdot \tilde{x} \leq h\}, \quad H = \begin{bmatrix} U \\ -L \end{bmatrix}, \quad h = \begin{bmatrix} \tilde{b} \\ -\tilde{a} \end{bmatrix} \tag{4-8}$$

Where

$$\begin{aligned} U &= ((G_{B_u} \boxtimes \mathbf{1}_p) - (\mathbf{1}_m \boxtimes I_p)) \cdot C, \quad \tilde{b} = \text{vec}\left(\tilde{B}_u\right) \\ L &= ((G_{A_u} \boxtimes \mathbf{1}_m) - (\mathbf{1}_n \boxtimes I_m)) \cdot G_{B_\theta} \cdot C, \tilde{a} = \text{vec}\left(\tilde{A}_u\right) \end{aligned} \tag{4-9}$$

The proof can be found in [6]. Note that any constraints with both time and quantity states can be eliminated. This is because their upper bound in $\tilde{b}$ will be ε and their lower bound in $\tilde{a}$ will be $\top$. This is a result of the structure of the block diagonal matrices $\tilde{B}_\theta$ and $\tilde{A}_\theta$.

This linearised system can be used to investigate whether the original system is bounded buffer stable. This will be done using stability criteria for conventional systems. This means that the linearised system for $\theta \in \{1, \ldots, S\}$ is bounded buffer stable if M_θ has multiplicative eigenvalues of less than or equal to 1, and all Jordan blocks of multiplicative eigenvalues of one are 1×1. This also means that the linearization is not bounded buffer stable if one multiplicative eigenvalue is larger than one or the Jordan block of the multiplicative eigenvalue of magnitude one is larger than 1×1 [6].
Since v^* is shift invariant in the direction of s, it means that M_θ has at least one multiplicative eigenvalue equal to one. Thus, the states of the system will not always converge back to the equilibrium point $\mathbf{0}$. If it is stable, the states will also not keep growing and will not diverge from each other. This has as a result that the Hilbert projective norm will be bounded for a stable linearised system. The MMPS system is bounded buffer stable at the temporal growth rate λ_θ within the region Ω_θ. Additionally, a stable linearised system ensures that none of the states grows unbounded, meaning the quantity states $\tilde{x}_q(k)$ remain bounded. From the transformation between the normalised state and actual state, it follows that if $\tilde{x}_q(k)$ is bounded, then $x_q(k)$ is also bounded.

4-3 Bounded Buffer Stability of Implicit MMPS Systems

Implicit MMPS systems are more complex than explicit ones due to additional dependency on the current state. This Section focuses on the linearization and bounded buffer stability of implicit MMPS systems.

Take a solvable normalized implicit MMPS system with multiple growth rates λ_θ,

$$\tilde{x}_\theta(k) = \tilde{A}_\theta \otimes \left(\tilde{B}_\theta \otimes' (C \cdot \tilde{x}_\theta(k-1) + D \cdot \tilde{x}_\theta(k))\right) \tag{4-10}$$

for $\theta \in \{1, \ldots, S\}$, and $A_\theta \in \mathbb{R}^{n \times m}, B_\theta \in \mathbb{R}^{m \times p}$, $C, D \in \mathbb{R}^{p \times n}$ with both temporal and quantity states. Just like in the explicit case seen in Section 4-2, it is possible to linearise the implicit MMPS system as such;

Definition 4.4. *(Linearization of an implicit MMPS system [6])*
Any normalised implicit MMPS system can be linearised as follows:

$$\begin{aligned} \tilde{x}_\theta(k) &= M_\theta \cdot \tilde{x}_\theta(k-1) \\ M_\theta &= (I - M_1)^{-1} \cdot M_2 \\ M_1 &= G_{\mathrm{A}_\theta} \cdot G_{\mathrm{B}_\theta} \cdot D \\ M_2 &= G_{\mathrm{A}_\theta} \cdot G_{\mathrm{B}_\theta} \cdot C \end{aligned} \tag{4-11}$$

for all $\tilde{x}(k) \in \Omega_\theta,\ k \in \mathbb{Z}^+$. If the inverse $(I - M_1)^{-1}$ exists.

Luckily, this inverse always exists for any solvable implicit MMPS system [5]. Notice that here again the footprint matrices G_{A_θ} and G_{B_θ} appear. Both matrices have exactly one entry equal to one in each row.

Just like with the explicit system, Ω_θ is the region where the linearization is valid. This region, which is given by a set of linear inequalities as given below:

$$\begin{aligned} &\Omega_\theta = \{\tilde{x} | H \cdot \tilde{x} \leq h\}, \quad H = \begin{bmatrix} U \\ -L \end{bmatrix}, \quad h = \begin{bmatrix} \tilde{b} \\ -\tilde{a} \end{bmatrix} \\ &U = ((G_{\mathrm{B}_\theta} \boxtimes \mathbf{1}_p) - (\mathbf{1}_m \boxtimes I_p)) \cdot (C + D \cdot M_\theta) \\ &L = ((G_{\mathrm{A}_\theta} \boxtimes \mathbf{1}_m) - (\mathbf{1}_n \boxtimes I_m)) \cdot G_{\mathrm{B}_\theta} \cdot (C + D \cdot M_\theta) \\ &\tilde{b} = \mathrm{vec}\left(\tilde{B}_\theta\right), \quad \tilde{a} = \mathrm{vec}\left(\tilde{A}_\theta\right) \end{aligned} \tag{4-12}$$

The proof can be found in [5].

M_θ will have at least one eigenvalue equal to one, with eigenvector v_1. This makes sense as M_θ is the linearisation of the normalised Implicit MMPS system, and by normalising, the system has been made invariant in the direction of v_1. Just as with the explicit linearization, the implicit linearization can also be used to investigate whether the original system is bounded-buffer stable.

This will be done using stability criteria for conventional systems. This means that the linearised system for $\theta \in \{1, \ldots, S\}$ is bounded buffer stable if M_θ has multiplicative eigenvalues of less than or equal to 1, and all Jordan blocks of multiplicative eigenvalues of one are 1×1.

This also means that it is not locally Max-Plus bounded buffer stable if one multiplicative eigenvalue is larger than one or the Jordan block of the multiplicative eigenvalue of magnitude one is larger than 1×1 [6].
As previously stated, M_θ has at least one multiplicative eigenvalue equal to one. Thus, the states of the system will not always converge back to the equilibrium point $\mathbf{0}$. If it is stable, the states will also not keep growing and will not diverge from each other. This has as a result that the Hilbert projective norm will be bounded for a stable linearised system. The MMPS system is bounded buffer stable at the temporal growth rate λ_θ within the region Ω_θ. Additionally, a stable linearised system ensures that none of the states grows unbounded, meaning the quantity states $\tilde{x}_q(k)$ remain bounded. From the transformation between the normalised state and actual state, it follows that if $\tilde{x}_q(k)$ is bounded, then $x_q(k)$ is also bounded.

As was seen in Section 3-2, implicit MMPS systems can have multiple fixed points with the same growth rate. This also results in the system matrix M_θ being the same for all vectors in the fixed-point set, which corresponds to the growth rate λ_θ. Even though the system matrix is the same for the different fixed points x_{e_i}, it is not necessarily the case for the regions Ω_{θ_i}. It is, however, easy to obtain the correct region when changing between these fixed points. Start with the the region $\Omega_{\theta_i} := \{\tilde{x} \mid H \cdot \tilde{x} \leq h_i\}$ of fixed point x_{e_i}. Then for a different fixed point x_{e_j} from the set of feasible fixed points $\mathcal{V}$, which by definition belongs to the same growth rate, the region is given by $\Omega_{\theta_j} = \left\{\tilde{x} \mid H \cdot \tilde{x} \leq h_1 + H \cdot \left(x_{\mathrm{e_i}} - x_{\mathrm{e_j}}\right)\right\}$

4-4 Maximal Invariant Set of a Linearised MMPS Systems

The linearisations given in Section 4-2 and 4-3 are only valid if the state $\tilde{x}_\theta(k)$ lies within Ω_θ. However it is not guaranteed that if $\tilde{x}_\theta(k)$ lies in Ω_θ, that $\tilde{x}_\theta(k+1)$ does as well. This is a large motivator to find these invariant sets. Since then, by definition, the next state will always be inside Ω_θ and the linearisation remains valid. It is assumed that the linearisation of the system is stable. The set $\mathcal{O}_\infty$ denoted the largest sub set of Ω_θ, such that once $\tilde{x}(0) \in \mathcal{O}_\infty$, $\tilde{x}(k)$ remains there for all $k > 0$. The largest invariant set $\mathcal{O}_\infty$ has to be determined numerically. In order to find this set, first, the definition of the precursor set must be given;

Definition 4.5. *(Precursor set [7])*
The precursor set for an autonomous system denoted by $\mathcal{O}$ is given by:

$$\mathrm{Pre}(\mathcal{O}) = \{x \in \mathbb{R}^n : M_\theta \cdot x \in \mathcal{O}\} \tag{4-13}$$

This means that the precursor set of Ω_θ is given by:

$$\mathrm{Pre}\left(\Omega_\theta\right) := H \cdot M_\theta \cdot \tilde{x} \leq h \tag{4-14}$$

Then Algorithm 2 can iteratively approximate the largest invariant set of Ω_θ.

Algorithm 2 Maximal positive invariant set [5]

Input: M_θ, Ω_θ
Output: $\mathcal{O}_\infty$
$\mathcal{O}_0 \leftarrow \Omega_\theta, k \leftarrow -1$
Repeat
$k \leftarrow k+1$
$\mathcal{O}_{k+1} \leftarrow \mathrm{Pre}\,(\mathcal{O}_k) \cap \mathcal{O}_k$
Until: $\mathcal{O}_{k+1} = \mathcal{O}_k$
$\mathcal{O}_\infty \leftarrow \Omega_k$

It can happen that Algorithm 2 will not terminate in a finite number of steps. However, in most cases, both for explicit and implicit MMPS systems, the algorithm does provide a solution in a finite number of steps and results in the largest invariant set of the linearised system. Often it even is the entire set of Ω_θ [5].

Chapter 5

Scalable Analysis of MMPS Systems

In the previous chapters, all the necessary background for MMPS systems was introduced. This chapter builds on that foundation and focuses on a new contribution: a MILP algorithm specifically designed for implicit MMPS systems. The goal is to significantly reduce the computational complexity involved in analysing such systems. Section 5-1 starts with an introduction to MILP, explaining the basics and the key differences between MILP and LPP problems. Then, in Section 5-2, the core MILP algorithm is presented. The discussion begins with the case of explicit systems, for which the existing theory is available. From this point onward, the contributions of this thesis start by extending the algorithm to handle implicit MMPS systems. The main challenges that arise in this extension, as well as the strategy developed to address them, are discussed in Section 5-3. Finally, Section 5-4 brings everything together into a complete MILP-based method for analysing implicit MMPS systems. It begins with a preprocessing step to reduce the feasible search space, followed by the full recursive MILP algorithm. The chapter concludes with an analysis of the complete MILP algorithm for implicit MMPS systems, including a runtime comparison against the current state-of-the-art LPP approach.

5-1 Introduction to Mixed Integer Linear Programming Problems

In the field of optimisation, there are several classes of problems defined. Problem classes include, for example, linear, nonlinear, convex, and semidefinite programming problems. The class tells you something about the shape of the problem, which also tells you how complex solving it is and what techniques and algorithms are available for a specific class of problem. Linear programming problems are the easiest class of optimisation problems to solve. They require a linear objective function and linear constraints. These constraints can be a combination of inequality constraints and equality constraints. Mathematically, a linear programming problem is defined as follows:

Definition 5.1. *(Linear programming problem [8])*
A linear programming problem is defined by a linear objective function which needs to be

minimised, while respecting the linear constraints:

$$\begin{aligned} \min_{x} \quad & c^\top x \\ s.t. \quad & A \cdot x \leq b \\ & x \in \mathbb{R}^n \end{aligned} \tag{5-1}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

A solver is tasked with minimising the value given by $c^\top x$ by deciding the values in x, where the elements in x are called the decision variables. While, of course, simultaneously ensuring that $A \cdot x \leq b$ remains true at all times. Current solvers are so fast at solving LPPs, that LPP problems with thousands of variables can be solved in seconds, and even problems with millions of variables in minutes to hours.
Another class of problems are the class of integer linear programming problems. Here, both the objective function and the constraints are still linear; however, the decision variables are all integers. An example of this could be how many products you can fit in a specific box. This slightly changes Definition 5.1 to:

Definition 5.2. *(Integer linear programming problem [8])*
An integer linear programming problem is defined by a linear objective function which needs to be minimised, while respecting the linear constraints with integer decision variables:

$$\begin{aligned} \min_{x} \quad & c^\top x \\ s.t. \quad & A \cdot x \leq b \\ & x \in \mathbb{Z}^n \end{aligned} \tag{5-2}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

It is also possible that these integer variables are binary, so only 0 or 1. This often occurs within system modelling, where the binary variable represents activation conditions such as something being 'on' or 'off'. This can easily be added in the constraints by bounding the integer variable to the range of [0,1].
MILPs, as the name suggests, combine both continuous linear programming problems and integer linear programming problems into one. This has as an effect that the decision vector will consist of both integer and continuous values. This is extremely useful in examples like optimising systems with hybrid dynamics. It is also not hard to see how combining Definition 5.1 and 5.2 can lead to a definition for an MILP problem, which is done below:

Definition 5.3. *(Mixed integer Linear Programming problems [8])*
An MILP is defined by a linear objective function which needs to be minimised, while respecting the linear constraints with both continuous and integer decision variables:

$$\begin{aligned} \min_{x} \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x = \begin{bmatrix} x_\mathrm{r}^\top & x_\mathrm{i}^\top \end{bmatrix}^\top \\ & x_\mathrm{r} \in \mathbb{R}^{n_\mathrm{r}} \qquad x_\mathrm{i} \in \mathbb{Z}^{n_\mathrm{i}} \end{aligned} \tag{5-3}$$

where $A \in \mathbb{R}^{m \times (n_r + n_i)}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^{n_r + n_i}$

This looks very similar to Definition 5.1 and 5.2. However, one issue with integer optimisation is the difficulty of solving. Since the number of feasible integer assignments grows exponentially with the number of integer variables, solving mixed-integer linear programming becomes increasingly challenging as the size of the problem increases. In general, MILPs are classified as NP-hard problems, which means that there is no known algorithm that can solve all assignments efficiently [9]. As a result, finding a solution to an MILP can take much more time than an LPP, as LPPs can be solved in polynomial time [8]. Luckily, there are smart ways of searching through the search space to find a solution. For more details on the workings of MILP methods, searches, and algorithms, look into chapter 11 of [10]. Section 5-2 and 5-3 will dive deeper into the workings of an MILP algorithm for MMPS systems analysis.

5-2 MILP for Implicit MMPS Systems

Before jumping into how MILP is used for analysing MMPS systems, one might wonder why an optimisation-based approach is needed in the first place. When analysing MMPS systems, one of the key questions is: what are the possible eigenvalues, and what are the regions (or modes) the system can operate in? These questions matter, because they tell us how the system behaves in the long term and whether certain behaviours are stable or not, as was previously discussed in Chapter 3 and 4.

A brute-force way to answer this is to check every possible region and see if a valid eigenvector-eigenvalue pair exists there. This essentially means solving a separate linear optimisation problem for every possible operating region. The number of operating regions is given by the number of footprint matrix pairs, which is given by: [5]

$$\prod_{i=1}^{n} a_i \cdot \prod_{j=1}^{m} b_j \tag{5-4}$$

Where a_i denotes the number of finite elements in row i of matrix A and b_j denotes the number of finite elements in row j of matrix B. Notice that this scales quadratically when the system size increases, and so, it becomes very time-consuming and does not scale well with the system size. In theory, using the LPP approach presented in [5] gives you all possible answers, but it is not practical for larger systems.

Luckily, for topical systems, it is known that there is only one valid eigenvalue region. So instead of checking all the tiny regions one by one, is it possible to search the space in a way that zooms in on the correct region, cutting down unnecessary work. Instead of solving a bunch of LPPs, topical systems can be analysed with a single MILP. This is already shown in [4].

In analysing the behaviour of MMPS systems, a relevant theoretical concept is the concept of modes. Intuitively, modes capture which entries of the system matrices are effectively active at a given event cycle. The formal definition is as follows:

Definition 5.4. *(Mode of an MMPS system [11])*
Consider an MMPS system described by:

$$\begin{aligned} x(k) &= A \otimes (B \otimes' z(k)) \\ z(k) &= C \cdot x(k-1) + D \cdot x(k) \end{aligned} \tag{5-5}$$

Here, $x(k) = [x_1(k), \ldots, x_n(k)]^\top$ and $z(k) = [z_1(k), \ldots, z_p(k)]^\top$. The mode of the system at event k refers to which entries of the matrices A and B are actively used in computing each $x_i(k)$.

More specifically, for each i, the value of $x_i(k)$ is determined by some combination of indices j and l such that:

$$x_i(k) = [A]_{i,j} + [B]_{j,l} + z_l(k) \tag{5-6}$$

This combination (j, l) is what we call the mode for index i at time k. It tells us which "path" through the system is currently active for computing that particular state component. The full system mode at time k is the collection of all such active index pairs across all i.

To illustrate, suppose row i of A contains a max over three possible terms. At event k, only one of these terms actually attains the maximum and contributes to $x_i(k)$. Similarly, each row of B may involve a min over several candidates, but again only one determines the outcome. The resulting choice of active entries across all rows forms the system's mode at that event.

Example 5.1. *(Modes of an MMPS system)*
Consider an MMPS system, described by the following equations:

$$\begin{aligned} x_1(k) &= 5x_2(k-1) - 2x_3(k-1) \\ x_2(k) &= \min\left(x_1(k) + 5, 2x_2(k-1) + x_3(k-1) - 4\right) \\ x_3(k) &= \max\left(x_2(k-1) - 3, x_1(k) + x_3(k-1) + 6\right) \end{aligned} \tag{5-7}$$

This MMPS system consists of 4 different modes, where depending on the current state, the system evolves according to the dynamics of these modes. These 4 modes are given as follows:

$$\begin{aligned} x_1(k) &= 5x_2(k-1) - 2x_3(k-1) \\ x_2(k) &= x_1(k) + 5 \\ x_3(k) &= x_2(k-1) - 3 \end{aligned} \tag{5-8}$$

$$\begin{aligned} x_1(k) &= 5x_2(k-1) - 2x_3(k-1) \\ x_2(k) &= x_1(k) + 5 \\ x_3(k) &= x_1(k) + x_3(k-1) + 6 \end{aligned} \tag{5-9}$$

$$\begin{aligned} x_1(k) &= 5x_2(k-1) - 2x_3(k-1) \\ x_2(k) &= 2x_2(k-1) + x_3(k-1) - 4 \\ x_3(k) &= x_2(k-1) - 3 \end{aligned} \tag{5-10}$$

$$\begin{aligned} x_1(k) &= 5x_2(k-1) - 2x_3(k-1) \\ x_2(k) &= 2x_2(k-1) + x_3(k-1) - 4 \\ x_3(k) &= x_1(k) + x_3(k-1) + 6 \end{aligned} \tag{5-11}$$

When a system is stable, every state grows with the same amount, also known as the growth rate. This means that if a system is stable and in its stationary regime, the system will be in the same mode every successive event cycle. Such a mode is called a dominant mode. When looking for the fixed points of a system, one can do this by using these modes. It is known that a topical system will only have one fixed point and one growth rate [4]. Thus, it is known that there is only one mode possible where the definition of a fixed point and growth rate can be satisfied. To recall, the definition of a fixed point is given by:

Definition 5.5. *(Fixed point and growth rate of MMPS systems)*
The time-invariant MMPS system $x(k) = f(x(k), x(k-1))$, $x \in \mathcal{R}^n$ and $f : \mathcal{R}^n \to \mathcal{R}^n$ with both time and quantity state has a fixed point if there exists a $\lambda \in \mathbb{R}$ and a vector $v \in \mathbb{R}^n$ such that

$$f(v) = v + \lambda \begin{bmatrix} \mathbf{1}_{n_t} \\ \mathbf{0}_{n_q} \end{bmatrix} \tag{5-12}$$

Where n_t is the number of time states and n_q is the number of quantity states. Then λ is called the growth rate and v the fixed point of the system f.

The MILP exploits the fact that only one mode can sustain a growth rate. The next section explains how this works.

5-2-1 MILP for Explicit Topical MMPS Systems

Having introduced the concepts of MILP problems and modes, this Section now turns to the derivation of an MILP-based algorithm for determining the growth rate and fixed points of explicit topical MMPS systems. The central idea is to identify the dominant modes of operation that sustain a particular growth rate. More precisely, the goal is to determine the growth rate λ, the fixed point x, and the active entries in the matrices A and B that characterise the operating mode of the system. The growth rate is obtained by solving an optimisation problem that minimises λ subject to the system dynamics and the additional requirement that the system remains in a stationary regime. So, very generally, the problem which needs to be solved is:

$$\begin{aligned} & \min \lambda \\ \text{subject to} \quad & \text{System dynamics} \\ & \text{System in stationary regime} \end{aligned} \tag{5-13}$$

However, optimisation solvers are not able to easily handle the $\otimes$ and $\otimes'$ operations. That is why the problem must be formulated such that it allows us to use existing solvers. Begin by rewriting the MMPS system into the extended state MMPS system form as defined in Section 3-2:

$$\begin{aligned} x(k) &= A \otimes y(k) \\ y(k) &= B \otimes' z(k) \\ z(k) &= C \cdot x(k-1) \end{aligned} \tag{5-14}$$

In the stationary regime, the following must hold:

$$\begin{aligned} x_e + s\lambda &= A \otimes y_e \\ y_e &= B \otimes' z_e \\ z_e &= C \cdot x_e \end{aligned} \tag{5-15}$$

Where $s = \begin{bmatrix} \mathbf{1}_{n_t}^\top & \mathbf{0}_{n_q}^\top \end{bmatrix}^\top$.
The subscript e will be omitted from now on for ease of notation.
To express this using conventional affine constraints, the fact that only one entry per row of

A and B will be responsible for the max and min is used, which means there is only one active element.
To track this active element in A and B, binary variables are introduced, $p_{j,l} \in \{0,1\}$ and $q_{i,j} \in \{0,1\}$. Where:

- $p_{j,l} = 1$ indicates that entry in column l is responsible for the minimum in row j of B
- $q_{i,j} = 1$ indicates that entry in column j is responsible for the maximum in row i of A

$\top$ and ϵ can never be the minimum or maximum value, so every $p_{j,l}$ or $q_{i,j}$ that corresponds to $\top$ or ϵ in B or A is automatically equal to 0. As is already known, only one entry will be the minimum or maximum, so to ensure that only one $p_{j,l}$ and one $q_{i,j}$ are equal to one per row, the following constraints must be added:

$$-\sum_{l} p_{jl} \leq -1 \quad \forall j, \quad -\sum_{j} q_{ij} \leq -1 \quad \forall i \tag{5-16}$$

To understand all constraints corresponding to $y = B \otimes' z$, it is sufficient to only look at the j^{th} element of y, y_j. This can be written as

$$y_j = \min_{l} \left(B_{jl} + z_l \right) \tag{5-17}$$

Since y_j is defined as the minimum over all l;

$$B_{jl} + z_l \geq y_j \quad \forall l \tag{5-18}$$

Must be true, and that equality holds only for the active l^* that activates the minimum. In other words;

$$B_{jl^*} + z_{l^*} = y_j \tag{5-19}$$

defines the unique entry of row j of B that is responsible for the values of y_j. In order to encode this into an optimisation problem. We use the following inequalities:

$$B_{jl} + z_l \leq y_j \quad \forall l \tag{5-20}$$

This condition always holds for the minimising l^*, but it does not generally hold for any other $l \neq l^*$. Therefore, by introducing the binary variable $p_{j,l}$, the constraint can be relaxed for all $l \neq l^*$ as follows:

$$-y_j + z_l + Mp_{jl} \leq -B_{jl} + M \quad \forall l \tag{5-21}$$

Where M is a sufficiently large constant. Since p_{jl} is only equal to 1 for l^* this results in a equality constraint for B_{j,l^*} As:

$$\begin{aligned} B_{jl^*} + z_{l^*} &\leq y_j \\ B_{jl^*} + z_{l^*} &\geq y_j \end{aligned} \tag{5-22}$$

Both can only be true if the equality holds. But for all other l, the inequality is:

$$-y_j + z_l + Mp_{jl} \leq -B_{jl} + M \quad \forall l \tag{5-23}$$

Simplified to:

$$-y_j + z_l \leq -B_{jl} + M \quad \forall l \tag{5-24}$$

Which is always true if M is large enough. Of course, this needs to be done for all elements in y:

$$\begin{aligned} & y_j - z_l \leq B_{jl} && \forall j, l \\ & -y_j + z_l + Mp_{jl} \leq -B_{jl} + M && \forall j, l \end{aligned} \tag{5-25}$$

In conclusion, the first constraint ensures that $y_j \leq B_{j,l} + z_l$, which must always be true. The second constraint ensures that only the element responsible for the current mode is an equality constraint, which is ultimately the goal.

For $x + s\lambda = A \otimes y$, the exact same can be done; however, now the signs need to be flipped as it involves the maximum. This results in the following constraints:

$$\begin{aligned} -s\lambda - x_i + y_j \leq -A_{ij} \quad \forall i, j \\ s\lambda + x_i - y_j + Mq_{ij} \leq A_{ij} + M \quad \forall i, j \end{aligned} \tag{5-26}$$

Lastly, the constraints for $z = C \cdot x$ must be determined. This function is already in conventional algebra, and thus the constraint can easily be added. This results in the final optimisation problem [4]:

$$\begin{aligned} & \min_{x,y,z,p,q} \lambda \\ \text{subject to} \quad & y_j - z_l \leq B_{jl} \quad \forall j, l \\ & y_j + z_l + Mp_{jl} \leq -B_{jl} + M \quad \forall j, l \\ & -s\lambda - x_i + y_j \leq -A_{ij} \quad \forall i, j \\ & s\lambda + x_i - y_j + Mq_{ij} \leq A_{ij} + M \quad \forall i, j \\ & -\sum_l p_{jl} \leq -1 \quad \forall j, \quad -\sum_j q_{ij} \leq -1 \quad \forall i \\ & z = C \cdot x \end{aligned} \tag{5-27}$$

Where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^j$, $z \in \mathbb{R}^l$, $p \in \{0,1\}^{j \times l}$, $q \in \{0,1\}^{i \times j}$.

The solver gets the freedom to choose the state x and the mode by choosing p and q such that it returns a growth rate and a fixed point which correspond to that dominant mode. Notice that p and q represent which mode is active based on the locations of the ones in every row. Also, notice that this method does not rely on the fact that the system is topical. This means that this algorithm will also work on non-topical explicit MMPS systems. However, since non-topical explicit MMPS systems can have multiple eigenvalues and eigenvectors, and the algorithm only gives one answer, one can never be sure whether the dominant mode, fixed point and growth rate which was found is the only one or whether there are more to be found. This is an issue that will be tackled in Section 5-3.

5-2-2 MILP for Implicit MMPS Systems

The algorithm that was presented in the previous section only works for explicit MMPS systems. However, it can also be adapted to work for implicit systems. Again, these implicit systems might have multiple eigenvalues and eigenvectors, and the algorithm will only provides one.

The objective again is to find a growth rate and fixed point of the system, and as a bonus, find the mode of the system, which is responsible for this fixed point and growth rate. So what are x_e and λ in the in extended MMPS form:

$$\begin{aligned} x_e + s \cdot \lambda &= A \otimes y_e \\ y_e &= B \otimes' z_e \\ z_e &= C \cdot x_e + D \cdot (x_e + s \cdot \lambda) \end{aligned} \tag{5-28}$$

Finding the growth rate will be done by minimising it while subject to the system's dynamical constraints, as well as constraining the system to be in a stationary regime similar to the explicit case. However, now while (5-13) is subject to (5-28).

Again, due to simplification in notation, the subscript e will be omitted from now on.
In order to change the algorithm from Subsection 5-2-1, z is rewritten as:

$$z = (C + D) \cdot x + D \cdot s \cdot \lambda \tag{5-29}$$

Then let w be

$$w = (C + D) \cdot x \tag{5-30}$$

and let d be

$$d = D \cdot s \tag{5-31}$$

Then $z = w + d \cdot \lambda$. Which means that

$$y = B \otimes' (w + d \cdot \lambda) \tag{5-32}$$

As a result, the constraints related to the $\otimes'$ in the MILP change to:

$$\begin{aligned} y_j - d_l\lambda - w_l \leq B_{jl} \quad \forall j, l \\ y_j + d_l\lambda + w_l + Mp_{jl} \leq -B_{jl} + M \quad \forall j, l \end{aligned} \tag{5-33}$$

The constraint $x + s \cdot \lambda = A \otimes y$ is the same in both the explicit and implicit extended MMPS form. This means that the constraints related to $x + s \cdot \lambda = A \otimes y$, in the optimisation problem stay the same. The last thing to change has to do with z. Since z is no longer present in the current constraints, but has been changed to w. This results in $z = C \cdot x$ becoming $w = (C + D) \cdot x$. Resulting in the final optimisation problem:

$$\begin{aligned} &\min_{x,y,z,p,q} \lambda \\ \text{subject to} \quad & y_j - d_l\lambda - w_l \leq B_{jl} \quad \forall j, l \\ & y_j + d_l\lambda + w_l + Mp_{jl} \leq -B_{jl} + M \quad \forall j, l \\ & -s\lambda - x_i + y_j \leq -A_{ij} \quad \forall i, j \\ & s\lambda + x_i - y_j + Mq_{ij} \leq A_{ij} + M \quad \forall i, j \\ & -\sum_l p_{jl} \leq -1 \quad \forall j, \quad -\sum_j q_{ij} \leq -1 \quad \forall i \\ & w = (C + D) \cdot x \quad d = D \cdot s \end{aligned} \tag{5-34}$$

Where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^j$, $w \in \mathbb{R}^l$, $p \in \{0,1\}^{j \times l}$, $q \in \{0,1\}^{i \times j}$, notice that when $D = \mathbf{0}$, the algorithms are the same, meaning that this new algorithm works for all types of MMPS systems, both explicit and implicit, and also topical and non-topical.

5-3 Search Tree for Footprint Matrices

The current approach to finding all valid eigenmodes is to check each possible footprint matrix pair individually using a brute force LPP method. This works, but the downside is that the number of these checks grows rapidly with system size. For example, when analysing the URS, increasing the size of the system from 4 to 6 stations already pushes the number of LPPs from 64 to over 1000. This makes full analysis impractical for larger systems [5]. As is now known, the MILP proposed in Subsection 5-2-2 can only give one solution, but it can be much quicker to analyse a full system; it is not known if the system has been fully analysed using the MILP from Subsection 5-2-2.

To address this, a different approach is taken: instead of checking all possible modes one by one, we use the MILP to guide the search. Since a single MILP only returns one solution, an iterative approach is needed to fully analyse the system. This is done by modifying the constraints to block previously found modes. If the MILP remains feasible, a new mode is discovered.

By using a search tree that explores the space of possible footprint matrix pairs, the MILP can be systematically guided. Each branch of the tree excludes or includes a specific combination of active entries, allowing the algorithm to explore the mode space efficiently and find all possible growth rates, fixed points and dominant modes.

Figure 5-1 helps to visualize this concept. Here, the entire search space is visualised as a large rectangle; there are 3 feasible solutions to the MILP and so 3 dominant modes in the system. However, only 1 can be viewed as optimal by the MILP, which here is indicated by the red dot. If the MILP is run on this problem, the algorithm will find the red solution since it is the optimal one in the entire search space. If, however, the search space gets limited to only the blue region, then there is only one feasible solution present in the new sub-search space, which means that it will also be the optimal solution to the MILP, so the MILP will find it and return it. Thus giving us a new dominant mode, which otherwise would have been invisible to us. If, then again, the search space is changed, but now to the green region, the solution in that region will be found. And so by only looking at a part of the search space, it becomes possible to uncover all feasible solutions, which was our goal.

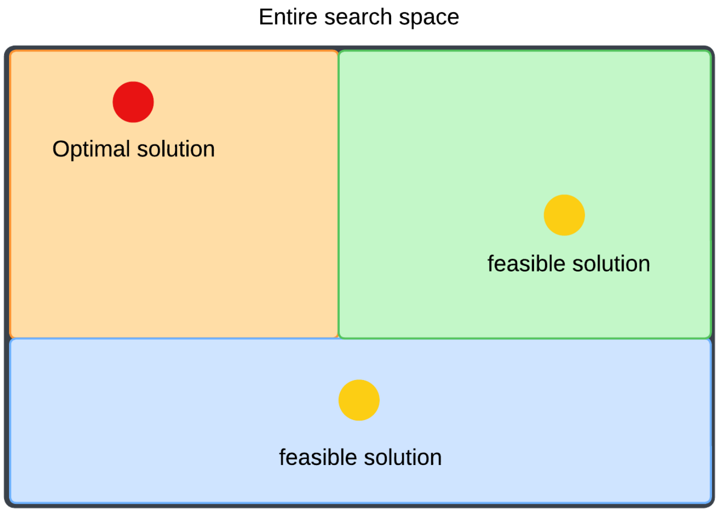


Figure 5-1: Example of partitioning the search space to find all feasible solutions

By changing the search space in a systematic way, the hope is to find all feasible solutions in a timely manner. Our search space can be seen as all the possible modes of the system. This means that it is possible to limit the search space of the MILP by placing extra constraints on p and q. As p and q tell which affine term of the MMPS function of a specific state is present in the dominant mode. By imposing that a specific affine term is present in the dominant mode, the search space is effectively restricted. In order to develop some intuition on how and why this works, let us first focus on only a single 3x3 p matrix. If the MILP finds a feasible solution, it will return a p matrix with exactly one 1 in every row. Since there is only 1 affine term from every MMPS function of every state active at any point in a cycle. This could, for example, lead to a p matrix of the form

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{5-35}$$

Again, remember that this 1 refers to that entry in B being in the dominant mode. However, as the aim is to know all dominant modes, let us consider all possible permutations of p. This can be visualised using a tree where every level of the tree corresponds to the same row in p, and every 'column' refers to the location where the 1 can be. So every path down is a permutation of the matrix p. The tree of a 3x3 matrix is visualised in Figure 5-2, notice that the red path corresponds to the matrix given in (5-35). The entire tree represents the search space of p.

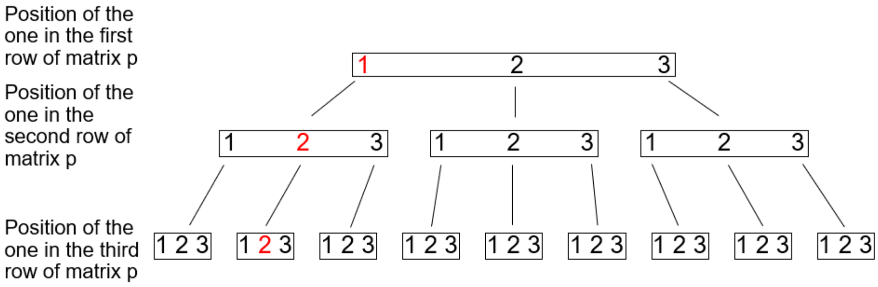


Figure 5-2: Tree representation of all possible permutations of a 3x3 p matrix

Now, by predetermining the location of the 1 in one of the rows. The first row in this case, due to ease of visualisation, it reduced the search space. So instead of giving the MILP a blank p matrix, one row is already filled in, like you can see in (5-36) and then see whether the MILP can find a feasible solution.

$$p = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \xrightarrow{\text{fix one row}} p = \begin{bmatrix} 0 & 1 & 0 \\ * & * & * \\ * & * & * \end{bmatrix} \tag{5-36}$$

This is equivalent to restricting the search space to only the blue area in Figure 5-3.

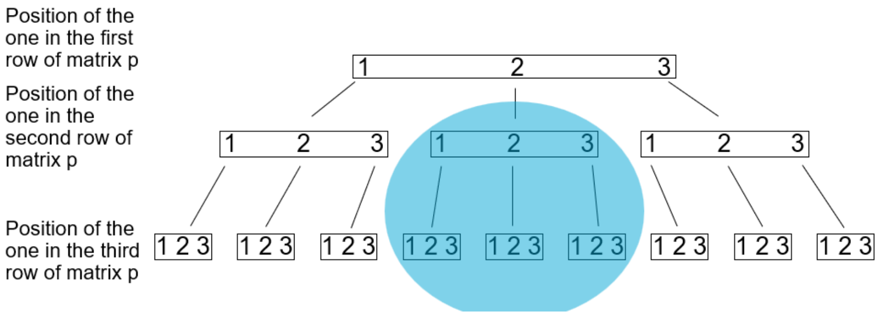


Figure 5-3: Example of restricting the search space of the MILP

Then, if there is a feasible solution, one must go one level deeper, or one row down, and fix the location of the active term in this new row. Since it is known that there is at least one solution in this new subspace. However, it is not certain if there are more. If there is no feasible solution, there is no need to explore the branch further since there is no solution in that entire sub-space. This means that the entire branch can be pruned and discarded. This would mean that one has eliminated, in this case, $\frac{1}{3}$ of the entire search space with just a single MILP call.
Notice that a standard MILP solver typically performs a similar process internally. This policy is now replicated and extended externally to systematically explore the entire search space, as the goal is to identify all feasible solutions. This section was designed to gain some intuition with regards to search trees, branches, pruning and the p and q matrices. In the next section, this principle will be used to design a smart search algorithm which will be combined with the MILP to create a full analysis algorithm for large, complex implicit MMPS systems.

5-4 MILP Search Tree Algorithm

As has been shown, the complexity of the problem grows quickly with the number of finite entries in the system matrices A and B, leading to an explosion in possible modes to check. While brute-force methods explore all footprint matrix pairs by solving an optimisation problem for each, this quickly becomes impractical for anything but small systems.

However, in practice, only a very small number of these modes can result in stationary behaviour, i.e. a fixed point. This means that a lot of finite entries in A and B are never present in any of the dominant modes, or a specific combination of affine terms is never possible together. This results in the LPP algorithm doing a lot of unnecessary work, slowing down analysis or even making it impossible due to the size. As described in Section 5-3, a search tree method can be used to explore, and most importantly, prune all possible modes more efficiently.

5-4-1 Preprocessing for Search Space Reduction

Before formulating the full MILP problem, it is useful to first reduce the size of the search space. The idea is that many of the affine terms appearing in the system matrices never play a role in any dominant mode, yet they still contribute to the complexity of the optimisation as they unnecessarily inflate the MILP search tree. By identifying and removing such terms in advance, the resulting MILP becomes significantly easier to solve.

The preprocessing step therefore, focuses on pruning away never dominant affine terms in A and B. Each affine term is tested individually by assuming its presence in a dominant mode and solving the MILP optimisation problem. If a feasible solution is found under this assumption, it indicates that the affine term is active in at least one dominant mode. If no feasible solution is found, it follows that in all modes where this affine term would appear, it can never be dominant. In that case, the affine term is discarded and replaced by either ϵ or $\top$, depending on whether it originates from A or B.

Importantly, one does not need to test any rows of A or B that only contain a single affine term. Since this term will always be active in any mode, and thus also in any dominant mode. This preprocessing is performed as described in Algorithm 3.

This preprocessing leaves us with only those affine terms that participate in at least one dominant mode. All other terms are excluded from further consideration in the MILP, resulting in a leaner and more efficient problem formulation. In other words, if $[B]_{jl}$ was infeasible, it is known that $p_{jl} = 0$, and if $[A]_{ij}$ was infeasible, then $q_{ij} = 0$ for all future analysis.

Preprocessing can be further accelerated by using the results from earlier steps. Since solving an MILP can be time-consuming due to the large number of integer variables, any reduction in the feasible search space will improve performance. If a particular affine term has already been identified as infeasible, meaning it cannot appear in any dominant mode, the corresponding binary variable can be fixed to zero in subsequent MILP calls. So for example, if it is known that $[A]_{1,2}$ will never be in a dominant mode, the variable $q_{1,2}$ can be set to zero for all next preprocessing steps since it is known that $[A]_{1,2}$ will never be in any dominant mode. This effectively reduces the number of free integer variables, which, as a result, speeds up each individual MILP solve and makes the overall preprocessing process more efficient.

Algorithm 3 Preprocessing Step for MILP Feasibility Analysis

for all (i, j) such that $[A]_{ij} \neq \varepsilon$ and $a_i \neq 1$ **do**
  Set $q_{ij} \leftarrow 1$
  Run MILP (5-34)
  if feasible **then**
    Store (i, j) with label A in feasible list
  else
    Store (i, j) with label A in infeasible list
  end if
end for
for all (j, ℓ) such that $[B]_{j\ell} \neq \top$ and $b_j \neq 1$ **do**
  Set $p_{j\ell} \leftarrow 1$
  Run MILP (5-34)
  if feasible **then**
    Store (j, ℓ) with label B in feasible list
  else
    Store (j, ℓ) with label B in infeasible list
  end if
end for

5-4-2 Recursive MILP Search Strategy

Now that there is a list of all affine terms that are present in at least one dominant mode, all these modes must be identified. This is achieved by combining the MILP algorithm of (5-34) and the tree search from Section 5-3. Before constructing the search tree, rows containing only a single finite entry are removed from this list, as that entry is guaranteed to be active in all modes. This includes the rows that contain a single finite term by design, but also the rows where preprocessing showed that only one affine term ever appears in a dominant mode, even if the row initially had multiple finite terms. Thus, $p_{j,l}$ or $p_{i,j}$ can be fixed for these rows. The remaining rows are combined into a table that is sorted in descending order based on the number of finite entries per row. Later on in this Section, it will be explained why this is better than an ascending order. This can, for example, look something like:

Number of column indices	RowIndex	ColumnIndices	Source
4	5	[6,7,8,9]	B
4	12	[21,22,23,24]	B
3	8	[8,9,10]	A
2	14	[31,32]	B
2	18	[45,46]	A

Table 5-1: Example of a search path table for dominant mode exploration

Table 5-1 is the basis of the search route to all feasible solutions, as all possible dominant modes are contained within these rows and columns. By choosing one column index for each row to equal 1, you end up with a complete p and q matrix. To explore all possible combinations, each row is treated as a branching point. This is the same as in Section 5-3, however, now with both p and q combined in a single tree. It is not needed to check every combination individually, but also not efficient to do so. For the analysis and optimisation, a depth-first search method will be used. A depth-first search has several advantages in this context:

- Memory efficiency: A depth-first search is more efficient in memory usage, which is beneficial as the systems at play are already large, so any memory-saving method is a win.
- Natural recursion: Depth-first search lends itself very well for recursive implementation. This is good since it allows for easy implementation and easy backtracking in case of infeasibility.

The full search algorithm can be found in pseudo code in Algorithm 4, and is explained in a more easily readable manner below:

1. Initialise by selecting the first row of the table. Let v be the current row index and w the first column index associated with v.
2. Depending on the source set, add one of the following constraints to the MILP:

$$\text{either} \quad q_{vw} = 1 \quad \text{if source is } A, \quad \text{or} \quad p_{vw} = 1 \quad \text{if source is } B$$

3. Solve the MILP.
   - **If the MILP is infeasible:**
     1. Remove the pair (v, w) from the table.
     2. Remove the added constraint.
     3. Move to the next column index w' associated with the same row v, and repeat from step 2.
     4. If all column indices for row v have been tried, backtrack to the previous row and continue with its next untried column index.
   - **If the MILP is feasible:**
     1. Proceed to the next row in the table.
     2. Repeat from step 1 with the new row index.
4. Continue this process until all feasible solutions are found. i.e. the entire tree is explored

With this new analysis strategy, instead of checking every possible mode to see if it is dominant, the process becomes more focused. It starts by finding modes that are already known to be dominant and then, in a sense, explores the neighbourhood around them to discover additional dominant modes. One might wonder if, with this neighbourhood search, all dominant modes are found and nothing is missed. However, this method guarantees that all dominant modes are found since the preprocessing only eliminates affine terms individually, which are never in a dominant mode. The full search checks every valid configuration involving the remaining affine terms. Only if a combination of several affine terms does not lead to a valid solution is it removed. But this is only the subset that includes that combination, so the rest is left in the search space. This ensures that no dominant modes are missed. To make the search process more efficient, the table of candidate rows is ordered from the most to the fewest available column indices. In other words, the rows with the most freedom (i.e., most possible active entries) are handled first. This has two advantages.

First, it allows the algorithm to prune large parts of the search tree early on. Since these high-flexibility rows contribute the most to the total number of mode combinations, restricting them first eliminates many potential branches at once.

Second, adding constraints for these rows early reduces the number of binary variables in the MILP more quickly. This not only shrinks the size of the problem, but also simplifies the remaining search steps, making each MILP solve faster and more tractable.

In the worst-case scenario, where every remaining combination leads to a feasible solution, this strategy no longer saves MILP calls. That is because then every branch and leaf must be visited, and every branch and leaf visited equates to an MILP call. In such cases, ordering the rows in ascending order would result in a shallower tree with fewer total leaves, making the full analysis cheaper. However, real-world systems typically only have a limited number of dominant modes, meaning many branches will be infeasible and thus pruned. Therefore, sorting in descending order remains the smart and efficient strategy for exploring the solution space.

Let us take a look at a numerical example to see how the algorithm works:

Example 5.2. *(MILP search tree algorithm in practice)*
Consider the linear time-invariant MMPS system given by (5-37) *taken from [11]. This is a non-topical MMPS system with 2 different growth rates and 5 different dominant modes. This example will show numerically how the new proposed MILP search algorithm works and how it is able to find all these dominant modes within a fraction of the time the LPP algorithm needs.*

$$\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \underbrace{\begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 3.5 \end{bmatrix}}_{A} \otimes \left(\underbrace{\begin{bmatrix} 0 & 10 & \top & \top \\ \top & 1.5 & \top & \top \\ \top & \top & 1.5 & 11.5 \\ \top & \top & \top & 1.5 \\ \top & 0 & \top & \top \\ \top & \top & \top & 0 \end{bmatrix}}_{B} \otimes' \left(\underbrace{\begin{bmatrix} -1 & 2 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}}_{C} \cdot \begin{bmatrix} x_1(k-1) \\ x_2(k-1) \end{bmatrix} \right) \right) \tag{5-37}$$

First, notice that there are 4 rows with more than one finite entry in A and B combined: rows A1, A2, B1 and B3. With 2, 6, 2 and 2 finite terms respectively, resulting in 48 different

footprint matrix pairs and thus 48 different modes.
Start by performing the preprocessing set, which results in 12 MILP calls, one for every affine term in a row with more than one affine term. This reveals that $[A]_{1,6}$, $[A]_{2,2}$, $[A]_{2,3}$, $[A]_{2,4}$, $[A]_{2,6}$ *are never present in an dominant mode. This means that only a fraction of the 48 different modes remain as candidates for being dominant. The search path table can be seen in Table 5-2, where it is clear there are now 6 modes left which have the potential to be dominant.*

Number of column indices	RowIndex	ColumnIndices	Source
2	1	[1,2]	B
2	2	[1,5]	A
2	3	[3,4]	B

Table 5-2: Search path table for dominant mode after preprocessing

After running the main search algorithm, 5 dominant modes are obtained. This gets achieved after running the MILP solver 10 times. This resulted in the following dominant modes.

$$\textit{for } \lambda = 2 : G_{A_1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} G_{B_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-38}$$

$$\textit{for } \lambda = 2 : G_{A_2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, G_{B_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-39}$$

$$\textit{for } \lambda = 10 : G_{A_3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, G_{B_3} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-40}$$

$$\textit{for } \lambda = 10 : G_{A_4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, G_{B_4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-41}$$

$$
\textit{for } \lambda = 10 : G_{A_5} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, G_{B_5} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-42}
$$

This entire analysis was done in 0.218 seconds by performing a total of 22 MILP calls as opposed to the otherwise required 48 LPP calls, which took 5.4 seconds.

5-4-3 Analysis of MILP Method

Now that the MILP-based search tree algorithm has been introduced, it is important to understand how it compares to the existing brute-force method in terms of computational effort. While the proposed approach can drastically reduce the number of optimisation calls needed, it is not always guaranteed to be faster overall.

This is mainly because of the fundamental difference in complexity between the two methods: solving an LPP is fast and reliable, while solving an MILP is significantly more expensive. This section breaks down the performance of the MILP method, highlighting both best- and worst-case scenarios, and shows under which conditions it offers a clear advantage. Additionally, we look at the solver used to run the MILPs, explaining why it was chosen and how it performs in practice. A series of example systems is used to compare the actual runtime of both methods, giving a more practical sense of the trade-offs involved.

For the new proposed strategy, the runtime is given by two parts. The preprocessing and the tree search. The prepossessing will require running:

$$
\text{number MILP calls during preprocessing} = \sum_{i=1}^{n} a_i + \sum_{j=1}^{m} b_j \tag{5-43}
$$

Where again a_i denotes the number of finite elements in row i of matrix A and b_j denotes the number of finite elements in row j of matrix B. In the worst case, all finite elements of A and B are present in a dominant mode, which means that the prepossessing has yielded no reduction in the number of MILPs which need to be evaluated by the algorithm. Then the MILP algorithm is run, and since there was no reduction in the tree since it is known that every element of A and B is present in a dominant mode. This means that every leaf of the tree must be evaluated since no branches can be pruned. The number of MILPs here depends on the construction of the tree, as described in the previous section. Again, a descending order in terms of the number of column indices is used. Combine a_i and b_j and sort these values in descending order into the list $\{n_1, n_2, \ldots, n_k\}$.

Then the total number of leaves and thus MILP calls is given by:

$$
N = \sum_{l=2}^{k} \prod_{m=1}^{l} n_m \tag{5-44}
$$

For example, if $\{n_1, n_2, n_3\} = \{4, 3, 2\}$, then:

$$N = (3 \times 4) + (2 \times 3 \times 4) = 36.$$

Combining this with the preprocessing step results in the worst-case total number of MILP calls needed:

$$\text{maximum number of MILP calls} = \sum_{i=1}^{n} a_i + \sum_{j=1}^{m} b_j + \sum_{l=2}^{k} \prod_{m=1}^{l} n_m. \tag{5-45}$$

The number of LPP calls is given by:

$$\text{number of LPP calls} = \prod_{i=1}^{n} a_i \cdot \prod_{j=1}^{m} b_j \tag{5-46}$$

Comparing the two shows that in the worst case, the number of LPPs is much less than the number of MILPs, which is not ideal. This makes sense, however, since the LPP strategy only checks final footprint matrix pairs, and the MILP also checks sub-filled pairs in the form of free variables in p and q. Remember that this worst case is where there is no reduction in the number of candidate dominant modes. This is not a realistic assumption to make, as in practice, there are a lot of affine terms which will never be in a dominant mode. In the best case, only the preprocessing step is needed, as that reveals that there is only one dominant mode, and thus the number of MILPs will be given by:

$$\sum_{i=1}^{n} a_i + \sum_{j=1}^{m} b_j \tag{5-47}$$

Which looks very similar to the number of LPPs given in (5-4). However, the product has become a summation, which makes the total number of calls much less, especially for larger systems. The actual reduction due to the preprocessing can not be known a priori, as the number of dominant modes is not known a priori. As a result, only after the preprocessing can one know the reduction. When this reduction is not significant, the MILP algorithm can have a much longer run time than the original LPP algorithm. This can easily be seen by the maximum number of MILP calls given by (5-45). One noteworthy aspect of the MILP algorithm, is that as you go deeper into the search tree, the number of free integer variables decreases. This reduction simplifies the MILP, causing it to behave more like an LPP, which leads to faster solution times. If the reduction in the search space due to the preprocessing is not significant enough, one can consider using a hybrid approach. This is where one combined the preprocessing with the current LPP strategy. After the preprocessing, all potentially feasible modes are checked using a set of LPPs instead of employing the MILP search algorithm. This will be beneficial since when the preprocessing yields no significant reduction in the number of potentially dominant modes, the number of MILPs which need to be called is similar to the number of LPPs. As is known, MILPs of similar size to LPPs generally have a longer run time. Again, this can only be beneficial if there are a lot of dominant modes and the preprocessing reduction is very limited.

The full MILP search tree algorithm, including preprocessing, was implemented using MATLAB. As a solver, Mosek was used. Mosek was chosen as it has an easy integration with

MATLAB, has a free educational license, but most importantly, Mosek is very good at handling large order differences in its constraints. Where other solvers generally like variables to be between 10^{-2} and 10^4, so an order difference of 10^6, Mosek has no issue in handling order differences of up to $10^{10} - 10^{12}$. This is an advantage since the systems used in Chapter 7 and 8 have these large order differences in their system equations, which are translated to large order differences in the constraints of the optimisation problems when analysing the systems.

A time comparison is made between using the current LPP method and the new proposed MILP method. Example 5.2, the URS from [5] and the transportation system from Chapter 8 serve as the benchmark. Both methods use the same solver, Mosek, to ensure a valid comparison. The results can be found in Table 5-3. It is very clear to see that for all three systems, the new MILP method is much faster.

	LPP (Solver: Mosek)	MILP (Solver: Mosek)
Example 5.2	5.4 sec	0.2 sec
URS(4 stations)	12.7 sec	0.9 sec
TPS	>25 years	2.5 hours

Table 5-3: Runtime comparison between the current LPP analysis method and the new MILP method on the URS and the Transportation System (TPS)

Algorithm 4 Dominant Mode Identification via MILP Search

Require: System matrices A, B, C, D and state classification vector s
Perform preprocessing as described by Algorithm 3
for all $[A]_{i,j}$ in infeasible list **do**
  Set $q_{ij} \leftarrow 0$
end for
for all $[B]_{j,\ell}$ in infeasible list **do**
  Set $p_{j\ell} \leftarrow 0$
end for
Construct table `search_path` with feasible list:
  (Number of column indices, row index, column candidates, source)
Sort `search_path` by descending number of column indices
Call DFS-Search(`search_path`, 1)
function DFS-Search(`search_path`, `row` $= 1$)
  if `row` $>$ number of rows in `search_path` **then**
    return found dominant mode
  end if
  Let $v \leftarrow$ current row index in `search_path` at `row`
  Let `columns` $\leftarrow$ column candidates for row v
  for all w in `columns` **do**
    if source for (v, w) is A **then**
      Set $q_{vw} \leftarrow 1$
    else if source for (v, w) is B **then**
      Set $p_{vw} \leftarrow 1$
    end if
    Run MILP (5-34)
    if MILP is feasible **then**
      Call DFS-Search(`search_path`, `row` $+ 1$)
    else
      Remove constraint for (v, w)
      Remove pair (v, w) from `search_path`
      continue with next w
    end if
  end for
  return backtrack to previous row (if any)
end function

Chapter 6

Periodicity of MMPS Systems

This chapter investigates the periodicity of MMPS systems. As established in Chapter 3, MMPS systems can exhibit a growth rate with period 1, where successive states increase at the same rate in every cycle. This chapter focuses on systems with periods greater than 1. Section 6-1 introduces definitions for such systems, formally defines the concept of a periodic orbit, and presents examples. Section 6-2 reviews existing results by deriving upper bounds for the period length in max-plus systems, based on the system size and structure. From this point onward, the contributions of this thesis begin; an bound upper periodic bound is established for min-plus systems, and the more general case of periodic MMPS systems is addressed in Section 6-3. This section introduces an extended periodic form that enables the use of existing analysis methods, and further examines the normalised behaviour of periodic MMPS systems and the stability of their periodic orbits.

6-1 Periodicity in MMPS Systems

From MMPS systems, it is known that if the system is stable, successive states after some $k > K$ cycles will all grow with the same amount [4]. This is known as the growth rate. Some MMPS systems do not have such a growth rate, but can still be stable. Furthermore, some MMPS systems are stable with a certain growth rate, but they also have something more; they are periodic.

These systems are referred to as periodic MMPS systems. For example, let us say that the state of a system evolves as follows:

$$x(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, x(1) = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, x(2) = \begin{bmatrix} 5 \\ 6 \end{bmatrix}, x(3) = \begin{bmatrix} 9 \\ 9 \end{bmatrix}. \tag{6-1}$$

Observe that over 3 cycles, all states of this system have grown by 9. This means a growth rate of 9 over 3 cycles. However, this does not mean a growth rate of 3 per cycle as the system does not grow uniformly at each cycle. Even after some transient time, this behaviour can remain. This non-uniform yet eventually cyclic behaviour is a critical property of periodic MMPS systems. Such behaviour requires more analysis than other non-periodic MMPS systems. Instead of a fixed point, these points are called periodic points of an MMPS system, which are formally defined as:

Definition 6.1. *(Periodic point of an MMPS systems [12])*
Consider an MMPS system of the form

$$x(k) = f(x(k-1), x(k)) \tag{6-2}$$

Then the vector z is a periodic point of f if for some $\mu \in \mathbb{R}$

$$f^p(z(k-1), z(k), \ldots, z(k+p)) = z(k-1) + \mu\mathbf{1} \tag{6-3}$$

The smallest p that satisfies this condition is called the period of f and the periodic point $z(k-1)$

Notice that Definition 6.1 implies that $z(k+p) = z(k) + \mu\mathbf{1}$. This additive form will be used, as it clearly shows how periodic behaviour works in MMPS systems.

Once a periodic point is found, the whole sequence of states it produces by evolving the system repeatedly becomes important. These states are called the periodic orbit of the system. Which is defined as follows;

Definition 6.2. *(Periodic orbit of an MMPS system [12])*
The sequence z(k), $k = 0, 1, \ldots,$ with $z(0)$ being a periodic point of f and $z(k) = f(z(k-1), z(k))$, $0 \leq k \leq p$ is called the periodic orbit of f.

The existence of a periodic orbit in an MMPS system does not exclude the possibility of a fixed point. Depending on the initial state, the system may converge to either a fixed point or a periodic trajectory. This is very clearly visible in Example 6.1.

Example 6.1. *(Periodic MMPS system with $p = 1$ and $p > 1$)*
Consider an MMPS system of the form

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \min\left(\max\left(x_2(k)+1, 1\right), 2\right) \\ \max\left(\min\left(x_1(k)-1, 1\right), 0\right) \end{bmatrix} \tag{6-4}$$

Which in ABC form becomes

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & 0 \end{bmatrix} \otimes \left(\begin{bmatrix} 1 & 2 & \top & \top & \top & \top \\ \top & \top & 0 & \top & \top & \top \\ \top & \top & \top & -1 & 1 & \top \\ \top & \top & \top & \top & \top & 0 \end{bmatrix} \otimes' \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \right) \right) \tag{6-5}$$

Depending on the initial condition, this system converges to a periodic stable growth trajectory with a period of either 1 or 2.

- ***Periodic trajectory (period of 2):***

$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \longrightarrow \mathbf{x}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \longrightarrow \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \longrightarrow \mathbf{x}_3 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \longrightarrow \ldots$$

- ***Periodic trajectory (period of 1):***

$$\mathbf{x}_0 = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \longrightarrow \mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \longrightarrow \mathbf{x}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \longrightarrow \dots$$

6-2 Bounds on Periods in Max-Plus and Min-Plus Systems

The previous section introduced a general definition of periodicity in the context of MMPS systems, covering periodic points, orbits, and growth rates. While that formulation applies to all MMPS systems, this section focuses more on a specific case of MMPS systems; max-plus and min-plus linear systems, where additional algebraic properties allow for guarantees regarding the upper bound on the period of the system, which is particularly useful when analysing convergence and long-term behaviour.

For max-plus systems, an upper bound on the period length is given by [13]. Which is

$$\max_S \operatorname{lcm}(S) \tag{6-6}$$

where S is a subset of $\{1, 2, \dots, n\}$ such that $\sum\limits_{j \in S} j \leq n$, where n refers to the number of states present in the system.
Take for example $n = 7$, then the upper-bound is given by:

$$\max(\operatorname{lcm}(2,2,3), \operatorname{lcm}(3,4), \operatorname{lcm}(2,5), \operatorname{lcm}(2,3), \dots) = \operatorname{lcm}(3,4) = 12 \tag{6-7}$$

Meaning that a max-plus system with 7 states can have at most a period of 12. Since this upper bound depends on the system dimension, it is tighter for systems with fewer states. For instance, consider the following max-plus system with 3 states:

Example 6.2. *(Periodicity of max plus linear Systems)*
Take the max-plus linear system:

$$x(k) = \begin{bmatrix} \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & 0 \\ 1 & \epsilon & \epsilon \end{bmatrix} \otimes x(k-1) \tag{6-8}$$

where $x(k) \in \mathbb{R}^3$. This system has 3 states, so $n = 3$, meaning that the maximum cycle length is

$$\max(\operatorname{lcm}(1,2), \operatorname{lcm}(1), \operatorname{lcm}(2), \operatorname{lcm}(3)) = \operatorname{lcm}(3) = 3 \tag{6-9}$$

If one is to simulate this systems starting from $x(0) = [0,0,0]^\top$ the following state sequence is obtained:

$$x(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, x(1) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, x(2) = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, x(3) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{6-10}$$

In this case, it is evident that after three cycles, all states have increased by 1. This implies a growth rate of $\mu = 1$, with a period of 3, equal to the maximum possible period for a system of this size. Furthermore, there is no initial condition that leads to a periodic point with period 1, i.e. a fixed point. This can be confirmed by noting that the system only has a single mode, and both the LPP and MILP algorithms yield no feasible solution for a fixed point.

In Example 6.2, the right starting point was chosen to immediately start inside the periodic orbit. This is, of course, not always the case. It can take some cycles before the trajectory enters a periodic orbit. This transition period is called the transient time. Let $A \in \mathbb{R}_\epsilon^{n\times n}$ be an irreducible max-plus matrix with eigenvalue λ and period of $p = p(A)$. Then there exists an integer $t(A)$ which is called the transient time, such that

$$A^{\otimes(k+p)} = \lambda^{\otimes p} \otimes A^{\otimes k}, \quad \forall k \geq t(A). \tag{6-11}$$

This confirms that the state sequence eventually becomes periodic with a period length equal to p. Even if the initial state lies outside the periodic orbit, after a finite number of cycles, the system will converge to the periodic region [1]. For max-plus, the upper bound of the period is known. Similarly, for min-plus linear systems, an upper bound can also be determined. We now turn to a minor but useful observation, which also serves to mark the beginning of the core contributions that follow:

Proposition 1. *(Maximum period length of min-plus systems)*
The upper bound to the period of Min-Plus linear systems is given by

$$\max_S \operatorname{lcm}(S) \tag{6-12}$$

where S is a subset of $\{1, 2, \ldots, n\}$ such that $\sum\limits_{j\in S} j \leq n$

Proof. Max-plus and min-plus algebra are isomorphic to each other [14], which means that any result for max-plus algebra and systems is also valid for Min-Plus algebra and min-plus systems. Given that the upper bound for the period of max-plus linear systems is given by

$$\max_S \operatorname{lcm}(S) \tag{6-13}$$

where S is a subset of $\{1, 2, \ldots, n\}$ such that $\sum\limits_{j\in S} j \leq n$, then the same holds for min-plus linear systems. $\square$

Example 6.3. *(Periodic behaviour of a min-plus linear system)*
Take the min-plus linear system:

$$x(k+1) = \begin{bmatrix} \top & 0.5 & \top \\ -0.5 & \top & -1.5 \\ -1.5 & -0.5 & -0.5 \end{bmatrix} \otimes x(k) \tag{6-14}$$

where $x(k) \in \mathbb{R}^3$. This system has 3 states, so $n = 3$, meaning that the maximum cycle length is

$$\max(\operatorname{lcm}(1,2), \operatorname{lcm}(1), \operatorname{lcm}(2), \operatorname{lcm}(3)) = \operatorname{lcm}(3) = 3 \tag{6-15}$$

when simulating this system starting from $x(0) = [-1, 1.5, 0]^\top$ the following state sequence is obtained;

$$x(0) = \begin{bmatrix} -1 \\ -1.5 \\ 0 \end{bmatrix}, x(1) = \begin{bmatrix} -2 \\ -0.5 \\ -1 \end{bmatrix}, x(2) = \begin{bmatrix} 1 \\ -1.5 \\ -0.5 \end{bmatrix}, x(3) = \begin{bmatrix} -2 \\ -0.5 \\ -1 \end{bmatrix}. \tag{6-16}$$

Here, it is very clear that states $x(1)$ and $x(3)$ are equal to each other. Meaning that $\mu = 0$, and thus also the eigenvalue $\lambda = 0$. This system has a period equal to 2, which is 1 less than the maximum period length.

6-3 Periodic Behaviour in General implicit MMPS Systems

So far, a general definition of periodicity has been discussed as well as periodic bound in max-plus and min-plus systems. However, for general MMPS systems, the situation becomes more complex. In this section, we will consider implicit time-invariant MMPS systems of the form [5]:

$$x(k) = A \otimes (B \otimes' (C \cdot x(k-1) + D \cdot x(k))) \tag{6-17}$$

and focus on systems that exhibit periodic behaviour with a period $p > 1$. These systems satisfy:

$$x(k+p) = x(k) + \mu\mathbf{1} \tag{6-18}$$

for some periodic growth rate μ. While systems with a period of $p = 1$ are relatively easy to analyse, systems with longer periods tend to be much harder and more time-consuming to analyse.

To understand this, first recall Algorithm 1, the power algorithm. The power algorithm looks at the difference between two states in the system evolution, and when $x(r) = x(q) \otimes (c \cdot s)$ with $r < q$ and $c \in \mathbb{N}$ is true, one has found an eigenvalue and eigenvector. An important observation is that $x(r)$ and $x(q)$ are not required to be consecutive cycles. When they are not, the difference $r-q$ reveals the period of the system's eventual cyclic behaviour. However, the other issues with the power algorithm remain. Namely, the power algorithm relies heavily on the choice of starting point, which results in an unknown transient time and uncertainty regarding how many eigenvalues and associated periodic regimes can be detected.

The MILP algorithm from Chapter 5 and the LPP algorithm from [5] only handle systems with periods of 1. This is because the construction of the optimisation problems relies on the property that in a fixed point, all states grow with the same growth rate. But when the period of a system is more than 1, the growth per cycle per state is no longer fixed even once you enter a periodic orbit. That is, (5-28) is no longer valid. It is also not possible to know this deviation beforehand; thus, the current methods break down. Moreover, in a periodic orbit, the sequence of active system modes changes across cycles, which further invalidates the current methods. However, systems of the form (6-17) with periods greater than 1 can be rewritten in a new form that incorporates the period internally. This form is called the extended periodic ABCD form.

Definition 6.3. *(Extended periodic ABCD form)*
The extended periodic ABCD form rewrites a periodic MMPS system into an ABCD structure that internalises the periodic behaviour, capturing the system's dynamics over one full period.

$$\underbrace{\begin{bmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+p) \end{bmatrix}}_{\hat{x}(n)} = \underbrace{\begin{bmatrix} A & \epsilon & \dots & \epsilon \\ \epsilon & A & \dots & \epsilon \\ \vdots & & \ddots & \epsilon \\ \epsilon & \epsilon & \dots & A \end{bmatrix}}_{\hat{A}} \otimes \left(\underbrace{\begin{bmatrix} B & \top & \dots & \top \\ \top & B & \dots & \top \\ \vdots & & \ddots & \\ \top & \top & \dots & B \end{bmatrix}}_{\hat{B}} \otimes' \right.$$
$$\left. \left(\underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & C \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & & \ddots & \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}}_{\hat{C}} \cdot \underbrace{\begin{bmatrix} x(k-p+1) \\ x(k-p+2) \\ \vdots \\ x(k) \end{bmatrix}}_{\hat{x}(n-1)} + \underbrace{\begin{bmatrix} D & \mathbf{0} & \dots & \mathbf{0} \\ C & D & & \vdots \\ \mathbf{0} & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \dots & C & D \end{bmatrix}}_{\hat{D}} \cdot \underbrace{\begin{bmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+p) \end{bmatrix}}_{\hat{x}(n)} \right) \right) \tag{6-19}$$

This new system

$$\hat{x}(n) = \hat{A} \otimes \left(\hat{B} \otimes' (\hat{C} \cdot \hat{x}(n-1) + \hat{D} \cdot \hat{x}(n)) \right) \tag{6-20}$$

is an implicit MMPS system with period $p = 1$ so (6-18) changes to $\hat{x}(k+1) = \hat{x}(k) + \mu\mathbf{1}$.

Notice that by rewriting the system into this new form, time invariance is preserved. As every block row of $[\hat{C}\ \hat{D}]$ contains exactly one C and one D matrix of the original system. And since our original MMPS system is time invariant, so is the extended periodic ABCD form. Since the extended MMPS system has a period of 1, the currently available analysis techniques can be used to identify the growth rate and fixed point of the extended periodic MMPS system. This can be done by solving a set of LPPs as defined in [5] or by using the proposed MILP algorithm from Chapter 5. Then $\hat{\lambda}$ and $\hat{x}_e$ are the resulting growth rate and fixed point of the extended MMPS system. The average growth rate over a period of the original system becomes $\lambda = \hat{\lambda}/p$. Notice that $\hat{x}_e$ now captures the entire periodic orbit into a stacked state vector. This means that all states in the periodic orbit can be considered as a periodic point, so;

$$x_{\mathrm{e}} = \hat{x}_{\mathrm{e}_{k\cdot n:(k+1)\cdot n}} \quad \forall\, 0 \leq k < p \tag{6-21}$$

Where $k \in \mathbb{Z}^+$. After a periodic orbit is found using the extended periodic MMPS system, the associated dominant mode of the extended periodic system can be obtained. Since the extended system unfolds the original system over p cycles, the dominant mode can be decomposed into a sequence of modes corresponding to each cycle in the original system's period. These individual modes, when taken together, generate the periodic orbit, even though none of them may be dominant on their own. We refer to these modes as the semi-dominant mode of the original MMPS system.

Definition 6.4. *(Semi-dominant mode of periodic MMPS systems)*
Consider a periodic MMPS system with period p, meaning the system has at least one periodic orbit of length p.

*Within such a periodic orbit, several modes may be responsible for sustaining the system's periodic behaviour. We define a **semi-dominant mode** as any mode that is actively used during at least one cycle within the periodic orbit.*

Since the orbit has period p, it can involve at most p distinct modes over its full evolution. Therefore, a given periodic orbit can have at most p semi-dominant modes.

Obtaining a semi-dominant mode from an extended periodic MMPS system is fairly straightforward once the dominant mode has been identified. After computing the dominant mode, this mode can be associated with a pair of matrices, denoted either as:

- The structure matrices $G_{\hat{A}'} G_{\hat{B}'}$ from the LPP, or
- The corresponding binary MILP matrices $\hat{q}, \hat{p}$ from the MILP

These matrices are block-diagonal with p diagonal blocks, corresponding to each cycle in the period. That is

$$G_{\hat{A}} = \begin{bmatrix} G_A^{(1)} & 0 & \cdots & 0 \\ 0 & G_A^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & G_A^{(p)} \end{bmatrix}, \quad G_{\hat{B}} = \begin{bmatrix} G_B^{(1)} & 0 & \cdots & 0 \\ 0 & G_B^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & G_B^{(p)} \end{bmatrix} \tag{6-22}$$

Or similarly for the binary matrices

$$\hat{q} = \begin{bmatrix} q^{(1)} & 0 & \cdots & 0 \\ 0 & q^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & q^{(p)} \end{bmatrix}, \quad \hat{p} = \begin{bmatrix} p^{(1)} & 0 & \cdots & 0 \\ 0 & p^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p^{(p)} \end{bmatrix} \tag{6-23}$$

Each diagonal block $\left(q^{(i)}, p^{(i)}\right)$ or $\left(G_A^{(i)}, G_B^{(i)}\right)$ corresponds to the mode active at cycle i within the period of length p. So each diagonal block $\left(q^{(i)}, p^{(i)}\right)$ or $\left(G_A^{(i)}, G_B^{(i)}\right)$ corresponds to a semi-dominant mode in the original MMPS system.

6-3-1 Unknown Period Length of Periodic MMPS Systems

If the period p of an MMPS system is not known in advance, it is still possible to write the system in its extended periodic ABCD form. However, analysing the system becomes more complex.

In this case, one must assume a value for p and check whether the system has a periodic orbit of that length. Fortunately, for any chosen p, a single MILP call is enough to determine whether such a period exists. If the MILP is infeasible, then it is certain that no periodic orbit of length p exists. If the MILP is feasible, then at least one periodic orbit of that length or of a periodic orbit of a factor of that length does exist. In that case, the full system must be analysed in more detail, as described in Chapter 5.

There are two main strategies for checking different values of p:

- Check periods in increasing order ($p = 1, 2, 3, \ldots, p_{\max}$)
- Check periods in decreasing order from some maximum ($p = p_{\max}, p_{\max} - 1, p_{\max} - 2, \ldots, 1$)

Each strategy has its advantages and disadvantages, and they both relate to a key property of the extended periodic form, which is that if a system has a true period p, then all multiples of p will also yield valid fixed points and growth rates in the MILP.

For example, suppose a system has a period $p = 2$, and the system is checked for all $p \leq p_{\max} = 5$. Then:

- At $p = 2$, we get a correct fixed point v and growth rate λ_2.
- At $p = 4$, the MILP will again return the same fixed point v, but with growth rate $\lambda_4 = 2 \cdot \lambda_2$, which is just a scaled version of the real one, so it is a false positive.

This leads to an important distinction between the two checking strategies:

- **Increasing order:** By checking smaller periods first, it is possible to detect and filter out false positives at larger periods by comparing fixed points. If a fixed point repeats and its growth rate is a multiple of a smaller one, we know it is redundant.
- **Decreasing order:** If we go from large p down to a lower value of p, we risk accepting false positives before the real period is found. To reduce the number of MILP calls in this case, we can exploit a useful structure:

Definition 6.5. *(Minimal covering set under divisibility)*
Given a finite set of positive integers $S \subseteq \mathbb{N}$, a subset $M \subseteq S$ is called a minimal covering set under divisibility if;

- *For every $s \in S$, there exist an $m \in M$ such that $s \mid m$; that is, s exactly divides m.*
- *No element $m \in M$ divides another element $m' \in M$ with $m \neq m'$*

Using this concept, one can limit MILP calls to just the minimal covering set under divisibility M rather than all values in S. This significantly reduces computational effort. However, because this approach checks larger periods first, it may return fixed points with incorrect growth rates. As a result, each fixed point must be post-processed to check whether its growth rate is a multiple of a lower one and correct it accordingly.
This naturally leads to the issue of how far the search should extend, or equivalently, the maximum period length that needs to be considered. From [15], it is known that a maximum period $p(n)$ exists for n-dimensional Max-Min-Plus (MMP) systems; however, this bound has not yet been determined. Since no bound is known for MMP systems, there is consequently no bound known for MMPS systems either.

Closely related to the notion of maximum period length is the idea of transient time. Introduced in Section 6-2 for max-plus and min-plus systems, also for MMPS systems, we can apply this concept.

Definition 6.6. *(Transient time of an MMPS systems)*
Consider a periodic MMPS function of the form

$$x(k) = f(x(k-1), x(k)) \tag{6-24}$$

with a period of $p = p(f)$ *and eigenvalue* λ*. Then there exists an integer t(f) which is called the transient time, such that*

$$f^p(x(k), \ldots, x(k+p)) = x(k) \otimes \lambda^{\otimes p} \tag{6-25}$$

for all $k \geq t(f)$

This definition allows us to reason not only about the periodic behaviour of MMPS systems, but also about how long it takes to reach that behaviour. Although determining the exact transient time $t(f)$ is often difficult in practice, its existence provides a theoretical guarantee that beyond a certain point, the system exhibits clean periodic dynamics. This is particularly useful when analysing or designing systems that are expected to stabilise or repeat their behaviour after a finite number of steps.

6-3-2 Normalised Periodic MMPS Systems

In some cases, it is beneficial to simplify the analysis of an MMPS system by transforming it into a normalised form. [5] has shown how one can normalise an MMPS system with a period of 1. The normalisation transforms the systems into a system with a growth rate of $\lambda = 0$ and a fixed point of $v = \mathbf{0}$ as well. Chapter 3 shows in detail how this normalisation can be performed. Periodic MMPS systems can also be normalised using this method. Then, instead of using a fixed point to normalise the system, one has to use any one of the states in a periodic orbit and perform the normalisation steps. After this is complete, one is left with a normalised $\tilde{A}$ and $\tilde{B}$. Note that this is the original system and not the extended system. When simulating the normalised system, something interesting happens with the state evolution. Instead of every state equaling zero, every p^{th} state is equal to zero, where p is the period of the system. Consider an periodic MMPS system with period p periodic orbit $z(0), \ldots, z(p)$ of the form

$$x(k) = A \otimes (B \otimes' (C \cdot x(k-1) + D \cdot x(k))) \tag{6-26}$$

Normalise the system with respect to $z(i)$. The resulting system is given by

$$\tilde{x}(k) = \tilde{A}_i \otimes (\tilde{B}_i \otimes' (C \cdot \tilde{x}(k-1) + D \cdot \tilde{x}(k))) \tag{6-27}$$

The state evolution of a normalised MMPS system with period $p = 1$ is as follows:

$$\mathbf{0} = \tilde{A} \otimes \underbrace{(\tilde{B} \otimes' \overbrace{(C \cdot \mathbf{0} + D \cdot \mathbf{0})}^{\mathbf{0}})}_{\mathbf{0}} \tag{6-28}$$

However, this is not the case for periodical normalised systems. Here, the state evolution is such that only the first and the p^{th} state are equal to zero.

$$\begin{aligned} x(1) &= \tilde{A} \otimes (\tilde{B} \otimes' (C \cdot \mathbf{0} + D \cdot x(1))) \\ x(2) &= \tilde{A} \otimes (\tilde{B} \otimes' (C \cdot x(1) + D \cdot x(2))) \\ &\vdots \\ x(p) = \mathbf{0} &= \tilde{A} \otimes (\tilde{B} \otimes' (C \cdot x(p-1) + D \cdot \mathbf{0})) \end{aligned} \tag{6-29}$$

This occurs because normalising the system effectively removes the constant growth rate that adds the same value to every state in each cycle. However, for periodic MMPS systems, the growth per cycle varies; in some cycles a state will grow more than the growth rate λ and in some cycles, it will grow less the the growth rate λ. However, since all states increase by $\mu = \lambda \cdot p$ over p cycles, the normalisation effectively resets the state to zero after p cycles, or compensates for this growth if the initial state is non-zero.

When examining the obtained $\tilde{A}$ and $\tilde{B}$ one should notice that

$$[\tilde{B}]_{j,\ell} \geq 0 \text{ and } [\tilde{A}]_{i,j} \leq 0 \tag{6-30}$$

no longer hold per definition. Additionally, every row of $\tilde{A}$ and $\tilde{B}$ is not required to have at least one zero element anymore. On the contrary, the extended periodic ABCD form does maintain these properties. Take, for example, the system from Example 6.2. When one performs the normalization as described in Chapter 3 using the the periodic point $z(0) = [0,0,0]^\top$ and growth rate $\lambda = \frac{1}{3}$, one ends up with the following $\tilde{A}$ and $\tilde{B}$;

$$\tilde{A} = \begin{bmatrix} \epsilon & -\frac{1}{3} & \epsilon \\ \epsilon & \epsilon & -\frac{1}{3} \\ \frac{2}{3} & \epsilon & \epsilon \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} 0 & \top & \top \\ \top & 0 & \top \\ \top & \top & 0 \end{bmatrix} \tag{6-31}$$

Since Example 6.2 is a max-plus linear system, $\tilde{B}$ is a min-plus identity matrix, which makes sense since B is a min-plus identity matrix as well. It is very clearly visible that $\tilde{A}$ does not have at least one 0 in every row, and there are also entries larger than zeros present. If one however, normalises the extended periodic form of this same system, which has a period of 3 and a growth rate of $\mu = 1$, one does get a system with a system which abides (6-30) and has at least one element equal to zero for every row of A and B. $\tilde{\tilde{A}}$ and $\tilde{\tilde{B}}$ are given as follows with a fixed point of v $v = [0,0,0,0,0,1,0,1,1]^\top$:

$$\tilde{\tilde{A}} = \begin{bmatrix} \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 & \epsilon & \epsilon \end{bmatrix} \quad \tilde{\tilde{B}} = \begin{bmatrix} 0 & \top & \top & \top & \top & \top & \top & \top & \top \\ \top & 0 & \top & \top & \top & \top & \top & \top & \top \\ \top & \top & 0 & \top & \top & \top & \top & \top & \top \\ \top & \top & \top & 0 & \top & \top & \top & \top & \top \\ \top & \top & \top & \top & 0 & \top & \top & \top & \top \\ \top & \top & \top & \top & \top & 0 & \top & \top & \top \\ \top & \top & \top & \top & \top & \top & 0 & \top & \top \\ \top & \top & \top & \top & \top & \top & \top & 0 & \top \\ \top & \top & \top & \top & \top & \top & \top & \top & 0 \end{bmatrix} \tag{6-32}$$

This is fairly trivial for this example, since the system has only a single mode. Regardless of the ordering of the periodic points in the extended fixed-point state, the resulting normalised pair remains the same.

6-3-3 Stability of Periodic MMPS Systems

In this section, the bounded-buffer stability of a periodic MMPS system is examined. The concept of bounded-buffer stability was already discussed in detail in Chapter 4. The notion of boundedness with regard to the stability of DE systems refers to the buffer levels taking on a constant value on average. This should result in the systems not overflowing. To determine whether an MMPS system is bounded-buffer stable, with a given growth rate λ, the system must be linearised. The first step in linearisation is normalisation. This is done by normalising the extended periodic ABCD form in accordance with Chapter 3. After obtaining the normalised system, one can continue with the linearisation. The definition of a linearised system is given in Chapter 4, but is repeated below;

Definition 6.7. *(Linearising an MMPS system [5])*
A normalised implicit MMPS system can be transformed into a system in conventional algebra for all $\tilde{x}_\theta(k) \in \Omega_\theta, k \in N$ by using the following:

$$\begin{aligned} \tilde{x}_\theta(k) &= M_\theta \cdot \tilde{x}_\theta(k-1) \\ M_\theta &= (I - M_1)^{-1} \cdot M_2 \\ M_1 &= G_{\mathrm{A}_\theta} \cdot G_{\mathrm{B}_\theta} \cdot D \\ M_2 &= G_{\mathrm{A}_\theta} \cdot G_{\mathrm{B}_\theta} \cdot C \end{aligned} \tag{6-33}$$

if the inverse $(I - M)^{-1}$ exists.

The polyhedron Ω_θ is the region in which the linearisation is valid. The coming section will examine this polyhedron for extended periodic MMPS systems, as well as describe a method to determine whether the system is bounded buffer stable. Recall from Chapter 4 that an linearised MMPS system is;

- Bounded buffer stable if M_θ has multiplicative eigenvalues of less than or equal to 1, and all Jordan blocks of multiplicative eigenvalues of one are 1×1.
- Not bounded-buffer stable if one multiplicative eigenvalue is larger than one or the Jordan block of the multiplicative eigenvalue of magnitude one is larger than 1×1.

Proposition 2. *(Stability of extended periodic ABCD MMPS systems)*
Consider a periodic MMPS system in extended periodic ABCD form. Let $G_{A_{\theta_1}}$, $G_{B_{\theta_1}}$, be the footprint matrices of the first semi-dominant mode in the periodic orbit, corresponding to the growth rate θ and let C be the C matrix from the original system. Let W_{n1} be the bottom-left block in the first block column of the inverse of $M = I - M_1$.

Then, the bounded buffer stability of the periodic orbit of the system can be determined by computing the eigenvalues of the matrix:

$$W_{n1} \cdot G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot C \tag{6-34}$$

If all eigenvalues are less than or equal to one and all Jordan blocks corresponding to the magnitude one are 1×1*, the extended periodic system is bounded buffer stable.*

Proof. From Definition 6.3 it follows that both $G_{\tilde{A}_\theta}$ and $G_{\tilde{B}_\theta}$ are block diagonal matrices as described by (6-22). From Definition 6.3 the shape of $\hat{C}$ and $\hat{D}$ is also known. Thus M_1 and M_2 can easily be constructed, which are given by (6-35) and (6-36) respectively.

$$M_1 = G_{\hat{\mathrm{A}}_\theta} \cdot G_{\hat{\mathrm{B}}_\theta} \cdot \hat{D} = \begin{bmatrix} G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot D & 0 & \cdots & 0 \\ G_{A_{\theta_2}} \cdot G_{B_{\theta_2}} \cdot C & G_{A_{\theta_2}} \cdot G_{B_{\theta_2}} \cdot D & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & G_{A_{\theta_p}} \cdot G_{B_{\theta_p}} \cdot C & G_{A_{\theta_p}} \cdot G_{B_{\theta_p}} \cdot D \end{bmatrix} \tag{6-35}$$

$$M_2 = G_{\hat{\mathrm{A}}_\theta} \cdot G_{\hat{\mathrm{B}}_\theta} \cdot \hat{C} = \begin{bmatrix} 0 & \cdots & 0 & G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot C \\ 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 \end{bmatrix} \tag{6-36}$$

From Definition 6.7 it is clear that M_1 must be subtracted from an identity matrix and inverted in order to proceed with the linearisation. $(I - M_1)$ is visible in (6-37).

$$I - M_1 = G_{\hat{\mathrm{A}}_\theta} \cdot G_{\hat{\mathrm{B}}_\theta} \cdot \hat{D} = \begin{bmatrix} I - G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot D & 0 & \cdots & 0 \\ G_{A_{\theta_2}} \cdot G_{B_{\theta_2}} \cdot C & I - G_{A_{\theta_2}} \cdot G_{B_{\theta_2}} \cdot D & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & G_{A_{\theta_p}} \cdot G_{B_{\theta_p}} \cdot C & I - G_{A_{\theta_p}} \cdot G_{B_{\theta_p}} \cdot D \end{bmatrix} \tag{6-37}$$

Notice that this is a lower block diagonal matrix. From [16] it is known that the inverse of a lower block triangular matrix is also a lower block triangular matrix. For ease of notation $(I - M_1)$ is replaced with M as such:

$$M = \begin{bmatrix} V_{11} & 0 & \cdots & 0 \\ V_{21} & V_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ V_{n1} & \cdots & V_{n,n-1} & V_{nn} \end{bmatrix} \quad M^{-1} = \begin{bmatrix} W_{11} & 0 & \dots & 0 \\ W_{21} & W_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ W_{n1} & \dots & W_{n,n-1} & W_{nn} \end{bmatrix} \tag{6-38}$$

Consider a block diagonal matrix

$$Z = \begin{bmatrix} A & 0 \\ C & D \end{bmatrix} \tag{6-39}$$

Then the inverse of z is given by [16]:

$$z^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -D^{-1}CA^{-1} & D^{-1} \end{bmatrix} \tag{6-40}$$

Now suppose that $[C|D] = [V_{n1} \dots V_{n,n-1}|V_{nn}]$ and

$$A = \begin{bmatrix} V_{11} & 0 & \cdots & 0 \\ V_{21} & V_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ V_{n-1,1} & \cdots & V_{n-1,n-2} & V_{n-1n-1} \end{bmatrix} \tag{6-41}$$

Then $-D^{-1}CA^{-1}$ is given by:

$$V_{nn}^{-1} \begin{bmatrix} V_{n1} \dots V_{n,n-1} \end{bmatrix} \begin{bmatrix} W_{11} & 0 & \dots & 0 \\ W_{21} & W_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ W_{n-1,1} & \dots & W_{n-1,n-2} & W_{n-1n-1} \end{bmatrix} \tag{6-42}$$

This can then be generalised to an equation to determine the inverse of the first block column of M as such:

$$W_{n1} = -V_{nn}^{-1} \sum_{k=1}^{n-1} V_{nk} W_{k1} \tag{6-43}$$

When multiplying M^{-1} with M_2, one ends up with:

$$M_\theta = M^{-1} \cdot M_2 = \begin{bmatrix} 0 & \dots & 0 & W_{11} \cdot G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot C \\ 0 & \ddots & 0 & W_{21} \cdot G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot C \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & W_{n1} \cdot G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot C \end{bmatrix} \tag{6-44}$$

To check the stability of the linearised system, one must calculate the eigenvalues of M_θ. From [16], it is known that the eigenvalues of a block triangular matrix are equal to the union of the eigenvalues of the diagonal blocks. M_θ is an upper block triangular matrix with all diagonal blocks equal to zero, except the final block. This means that the stability of the extended periodic ABCD form can be determined by examining the eigenvalues of the matrix:

$$W_{n1} \cdot G_{A_{\theta_1}} \cdot G_{B_{\theta_1}} \cdot C \tag{6-45}$$

□

If one finds that the eigenvalues of (6-34) are less than or equal to 1, and all Jordan blocks of eigenvalues equal to 1 are 1×1, one can conclude that the extended periodic MMPS system is bounded-buffer stable. Determining the valid region for this linearised system can be done in the same manner as described in Chapter 4, and will thus not be discussed here. The same holds for the invariant set. To illustrate these concepts, the following example examines the eigenvalues of a linearised extended periodic MMPS system.

Example 6.4. *(Eigenvalues of linearised extended periodic MMPS system)*
Consider the system from Example 6.1 in extended periodic ABCD form. This system normalised with respect to $\hat{x}_e = [1, 1, 2, 0]^\top$*, results in the following* $\tilde{\hat{A}}$ *and* $\tilde{\hat{B}}$*:*

$$\tilde{\hat{A}} = \begin{bmatrix} 0 & 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & -1 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 & -1 & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 & 0 \end{bmatrix}$$

$$\tilde{\hat{B}} = \begin{bmatrix} 0 & 1 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\ \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top \\ \top & \top & \top & 0 & 0 & \top & \top & \top & \top & \top & \top & \top \\ \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top \\ \top & \top & \top & \top & \top & \top & 0 & 0 & \top & \top & \top & \top \\ \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top \\ \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & 1 & \top \\ \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 \end{bmatrix} \tag{6-46}$$

Then, in accordance with Proposition 2, linearising the system results in the following M_θ*:*

$$M_\theta = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-47}$$

Which is a block matrix with all blocks zero except the final block column. Where the stability of the linearisation only depends on the eigenvalues of the bottom right block. These eigenvalues are 1 and 1, both with a Jordan block of 1×1*, which means that this extended periodic MMPS system is bounded-buffer stable.*

It can happen that a periodic MMPS system has a state that blows up, and then is corrected in the next cycle. Such a state evolution could look something like;

$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \to \mathbf{x}_1 = \begin{bmatrix} \infty \\ 1 \end{bmatrix} \to \mathbf{x}_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \to \mathbf{x}_3 = \begin{bmatrix} \infty \\ 2 \end{bmatrix} \to \dots \tag{6-48}$$

The entire periodic orbit is stable, while internally, some modes are not. Therefore, it is also worth linearising the original system based on its semi-dominant modes. In doing so, one can determine if these semi-dominant modes are bounded-buffer stable by checking the Hilbert projective norm of the state.

$$\begin{aligned} \|x_{\mathrm{t}\theta}(k)\|_{\mathbb{P}} &= \|\tilde{x}_{\mathrm{t}\theta}(k-p) + x_{\mathrm{te}\theta} + \lambda_\theta \cdot k \cdot p \cdot 1\|_{\mathbb{P}} \\ &= \|\tilde{x}_{\mathrm{t}\theta}(k-p) + x_{\mathrm{te}\theta}\|_{\mathbb{P}} \leq \|\tilde{x}_{\mathrm{t}}(k)\|_{\mathbb{P}} + \|x_{\mathrm{te}\theta}\|_{\mathbb{P}} \end{aligned} \tag{6-49}$$

Which is bounded buffer stable if the eigenvalues of the linearised system are less than, or equal to 1, and all Jordan blocks of eigenvalues equal to 1 are 1×1. In this case, it is not guaranteed that the linearisation regions align or even overlap. This needs to be carefully verified for each region. If the regions are disconnected, determining a maximum invariant set around the periodic orbit becomes significantly more challenging, as transitions between the regions are no longer straightforward. In such cases, it must be proven that switching between regions actually occurs, and under what conditions, before any meaningful analysis of the invariant set can be done.

Chapter 7

Modelling Framework for Transportation Networks

This chapter introduces a way to turn a simple transport system into a recursive extended system. The main contributions are a general framework for modular MMPS sub-systems and a systematic way of turning a high-level system description in the form of an adjacency graph into a system of equations. Section 7-1 introduces the basics for modular MMPS sub-systems; what do they look like, and what properties should hold for time invariance and solvability. Furthermore, the Section dives into the scenario where there is no synchronisation within a system. Section 7-2 proposes a method how switching MMPS systems can be written as a single MMPS system. This is then used in Chapter 8 in practice on a real-world system. Finishing off the chapter are Section 7-3 and Section 7-4, which deal with a toolbox of nodes one can choose from in order to easily construct a system of equations of a transportation system. In Section 7-3, some different nodes are introduced and derived. Thereafter, some other more advanced nodes are briefly discussed but not derived. Section 7-4 then uses these nodes to construct a system of equations in ABCD form, from a high-level system description consisting of a graph together with the nodes' properties.

7-1 Basics of Modular Transportation Systems

Modelling a large transportation network is a time-consuming task. Deriving all the equations can be tedious, and transforming the equations into the ABCD form for easy simulation and analysis is prone to mistakes. Therefore, an easier way of modelling these systems should be developed.

Just like in the URS[5], a modular system is ideal to allow for more complex modelling, relying on patterns and other properties to reuse specific elements. However, unlike the URS, a transportation system often lacks a regular structure, making it difficult to simply stack identical nodes or matrices.

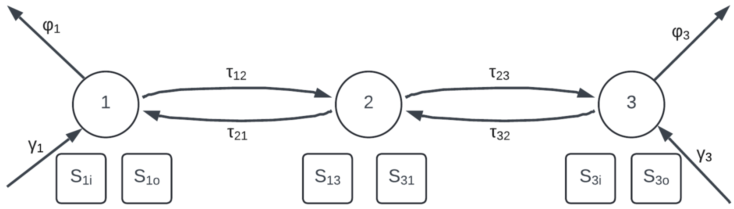


Figure 7-1: Example of a system with three nodes for modular modelling

This section presents a method for achieving a modular transportation system. Such a method will allow one to take sub-systems and easily connect them. This is useful in the case of parallel development, expanding existing systems or even the construction of completely new systems. This section will discuss a general way of modelling large transportation networks by relying on pre-constructed sub-systems, nodes and patterns. First, this will be done on a small node-based level, which will then be expanded to a subsystem level.

7-1-1 Introduction to Modular Transport Nodes

Before diving into modular sub-systems, the general idea will be presented using a very simple 3-node transportation system. Take the network seen in Figure 7-1. A vehicle drives between node 1 and 2, and node 2 and 3 delivering goods. In this system, there is an inflow of goods (γ_i) at the left and right nodes, also named 'end nodes', which needs to be delivered via the vehicles to the other end node, where they are delivered with an outflow of goods (φ_i). Each node has two stacks that can be used for storage while waiting to be picked up. Two stacks for each node, as this easily encodes the origin and destination of the goods. Knowing which flows take goods from which exact stack is not necessary.

In this small case, it is relatively easy to derive all the state equations. However, if the system becomes larger, this might not be the case. Notice, that even in this small system, there are already repeating nodes; i.e. nodes 1 and 3 have the same dynamics, they both have an in- and outflow of goods, two stacks, one vehicle delivering and picking up goods, and they have the same interactions and behaviour within the node. With larger systems, repeating nodes are even more common. The following question arises: how can this system be modelled more easily while keeping this repetition in mind?

Let us separate one node of this system, node 1 in this case, visible in Figure 7-2. The dynamical behaviour of this node remains unchanged regardless of whether it is directly connected to some other node. Now this node has an input u_{11} and an output y_{11}, where the subscript 11 refers to in- or output 1 of node 1. This input must contain all the information the dynamical equations need to give a unique result. The output y_{11} can be any state or combination of states, whichever is needed by either the user or a connected node. The exact dynamics of this node are not relevant. But for this node, we can define a state vector $x_1 \in \mathbb{R}^{n_1}$. The dynamics of this single node can be described by an MMPS function in

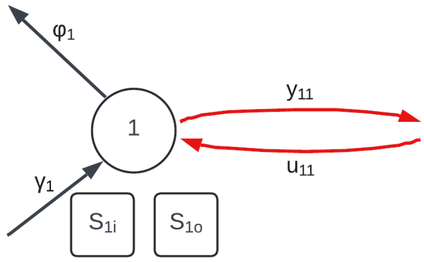


Figure 7-2: Separated node 1 from Figure 7-1

ABCDE form:

$$\begin{aligned}\begin{bmatrix} x_{1,\mathrm{t}}(k) \\ x_{1,\mathrm{q}}(k) \end{bmatrix} = \underbrace{\begin{bmatrix} A_{1,\mathrm{t}} & \varepsilon \\ \varepsilon & A_{1,\mathrm{q}} \end{bmatrix}}_{A} \otimes \left(\underbrace{\begin{bmatrix} B_{1,\mathrm{t}} & \top \\ \top & B_{1,\mathrm{q}} \end{bmatrix}}_{B} \otimes' \left(\underbrace{\begin{bmatrix} C_{1,11} & C_{1,12} \\ C_{1,21} & C_{1,22} \end{bmatrix}}_{C} \cdot \begin{bmatrix} x_{1,\mathrm{t}}(k-1) \\ x_{1,\mathrm{q}}(k-1) \end{bmatrix} \right.\right. \\ \left.\left. + \underbrace{\begin{bmatrix} D_{1,11} & D_{1,12} \\ D_{1,21} & D_{1,22} \end{bmatrix}}_{D} \cdot \begin{bmatrix} x_{1,t}(k) \\ x_{1,q}(k) \end{bmatrix} + \underbrace{\begin{bmatrix} E_{1,11} & E_{1,12} \\ E_{1,21} & E_{1,22} \end{bmatrix}}_{E} \cdot \begin{bmatrix} u_{11,t}(k) \\ u_{11,q}(k) \end{bmatrix} \right)\right)\end{aligned} \tag{7-1}$$

The output of this node, y, is specified through another MMPS function, whose form can be selected as desired, which is given by:

$$y_{11}(k) = F_1 \otimes (H_1 \otimes' (K_1 \cdot x_1(k-1) + L_1 \cdot x_1(k))) \tag{7-2}$$

Suppose that every node of the system in Figure 7-1 is modelled as an independent MMPS system, in such a way that the output of one node is exactly the input of the node it is connected to. Then one can simply 'connect' the blocks and obtain the system of equations that govern the system. To demonstrate this, the system from Figure 7-1 will be partitioned into three subsystems, as shown in Figure 7-3. Note that the subsystems are simply the individual nodes in this case.

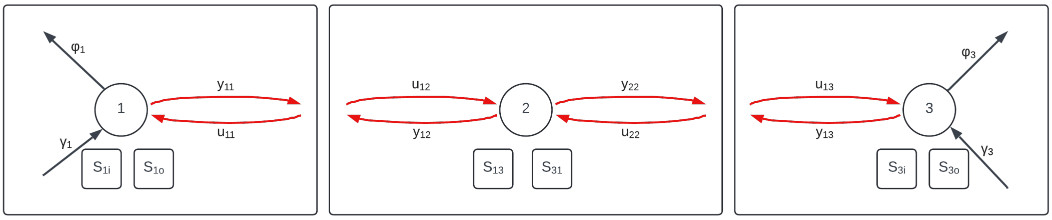


Figure 7-3: Partitioning of example system

For each node, one can derive the governing equations and model ever node individually as such:

$$
\begin{aligned}
x_1(k) &= A_1 \otimes (B_1 \otimes' (C_1 \cdot x_1(k-1) + D_1 \cdot x_1(k) + E_1 \cdot u_{11}(k))) \\
y_{11}(k) &= F_1 \otimes (H_1 \otimes' (K_1 \cdot x_1(k-1) + L_1 \cdot x_1(k))) \\
\\
x_2(k) &= A_2 \otimes (B_2 \otimes' (C_2 \cdot x_2(k-1) + D_2 \cdot x_2(k) + E_{12} \cdot u_{12}(k)) + E_{22} \cdot u_{22}(k))) \\
y_{12}(k) &= F_{12} \otimes (H_{12} \otimes' (K_{12} \cdot x_2(k-1) + L_{12} \cdot x_2(k))) \\
y_{22}(k) &= F_{22} \otimes (H_{22} \otimes' (K_{22} \cdot x_2(k-1) + L_{22} \cdot x_2(k))) \\
\\
x_3(k) &= A_3 \otimes (B_3 \otimes' (C_3 \cdot x_3(k-1) + D_3 \cdot x_3(k) + E_3 \cdot u_{13}(k))) \\
y_{13}(k) &= F_3 \otimes (H_3 \otimes' (K_3 \cdot x_3(k-1) + L_3 \cdot x_3(k)))
\end{aligned}
\tag{7-3}
$$

Again, the exact equations are not relevant; only the structure is. Since node 1 and node 3 are the same, it is known that A_1 and A_3 have the same structure, potentially different variables. Such as different travel times or capacities. This also holds for the B, C, D, E, F, H K and L matrices. These independent systems can be connected to obtain the original 3-node transportation system. The system appears to be an open-loop system; however, the outputs of one subsystem are the inputs of another, which means that this is actually a closed-loop system.

Combining the equations into the description of the entire system results in:

$$
\begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ y_{11}(k) \\ y_{12}(k) \\ y_{22}(k) \\ y_{13}(k) \end{bmatrix} =
\begin{bmatrix} A_1 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & A_2 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & A_3 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & F_1 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & F_{12} & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & F_{22} & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & F_3 \end{bmatrix}
\otimes \left(\begin{bmatrix} B_1 & \top & \top & \top & \top & \top & \top \\ \top & B_2 & \top & \top & \top & \top & \top \\ \top & \top & B_3 & \top & \top & \top & \top \\ \top & \top & \top & H_1 & \top & \top & \top \\ \top & \top & \top & \top & H_{12} & \top & \top \\ \top & \top & \top & \top & \top & H_{22} & \top \\ \top & \top & \top & \top & \top & \top & H_3 \end{bmatrix} \right.
$$

$$
\otimes' \left(\begin{bmatrix} C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C_3 & 0 & 0 & 0 & 0 \\ K_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{12} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & K_{13} & 0 & 0 & 0 & 0 \end{bmatrix}
\begin{bmatrix} x_1(k-1) \\ x_2(k-1) \\ x_3(k-1) \\ y_{11}(k-1) \\ y_{12}(k-1) \\ y_{22}(k-1) \\ y_{13}(k-1) \end{bmatrix} + \right.
$$

$$
\left. \left. \begin{bmatrix} D_1 & 0 & 0 & 0 & E_{11} & 0 & 0 \\ 0 & D_2 & 0 & E_{12} & 0 & 0 & E_{22} \\ 0 & 0 & D_3 & 0 & 0 & E_{13} & 0 \\ L_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & L_{12} & 0 & 0 & 0 & 0 & 0 \\ 0 & L_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & L_{13} & 0 & 0 & 0 & 0 \end{bmatrix}
\begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ y_{11}(k) \\ y_{12}(k) \\ y_{22}(k) \\ y_{13}(k) \end{bmatrix} \right) \right) \tag{7-4}
$$

Notice that the system of (7-3), which appeared to be open loop, is actually a closed loop once the nodes are reconnected, as is visible in (7-4). Additionally, notice that the state of the system has increased. This system was just a small example of a small system to grasp the idea. In the next section, a more formal way, that also works for larger subsystems, will be discussed.

7-1-2 Introduction to MMPS Sub-Systems

Large systems can be divided into smaller, interconnected sub-systems to simplify modelling and analysis. This section formalises how to represent and connect MMPS sub-systems, enabling efficient handling of complex networks. Key properties, like time invariance and solvability are discussed, along with how to describe both open- and closed-loop interconnected systems.

Instead of single nodes, MMPS sub-systems will be considered. An MMPS sub-system is a smaller, self-contained part of a larger MMPS system that can be modelled independently. When multiple sub-systems interact, they exchange information through defined inputs and outputs. By connecting these sub-systems, we can build complex systems while keeping the modelling and analysis manageable.

Take system 1 and system 2 visible in Figure 7-4. These systems can, for example, represent the logistics network in 2 different countries that have formed a partnership, and now they will be viewed as a single system. It is possible to reformulate all equations of the entire new

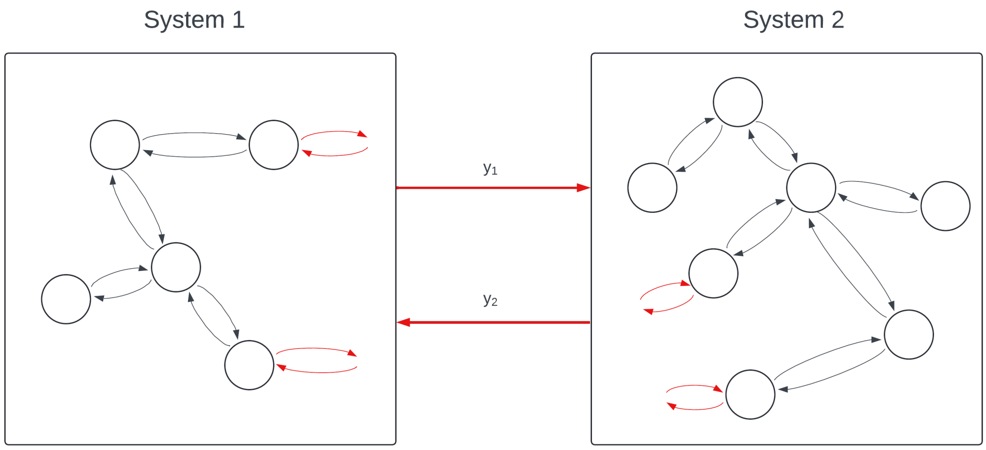


Figure 7-4: Example of two subsystems being connected

system, or something smarter can be done, as briefly introduced in Subsection 7-1-1. The red arrows y_1 and y_2 indicate some information which needs to be transferred between the two sub-systems. This can be a state from the sub-system or a combination of states.

Suppose that the original systems 1 and 2 can be described by:

$$\begin{aligned} x_1(k) &= f_1(x_1(k-1), x_1(k), u_1(k)) \\ x_2(k) &= f_2(x_2(k-1), x_2(k), u_2(k)) \end{aligned} \tag{7-5}$$

Where $x_1 \in \mathbb{R}^n$, $x_2 \in \mathbb{R}^m$, $u_1 \in \mathbb{R}^p$, $u_2 \in \mathbb{R}^q$, f_1 is an MMPS function $f_1 : \mathbb{R}^{n+p} \to \mathbb{R}^n$ and f_2 is an MMPS function $f_2 : \mathbb{R}^{m+q} \to \mathbb{R}^m$.

The information being sent between the systems, y_1 and y_2 is given by:

$$\begin{aligned} y_1(k) &= g_1(x_1(k-1), x_1(k)) \\ y_2(k) &= g_2(x_2(k-1), x_2(k)) \end{aligned} \tag{7-6}$$

Where g_1 is an MMPS function $g_1 : \mathbb{R}^{2n} \to \mathbb{R}^p$ and g_2 is an MMPS function $g_2 : \mathbb{R}^{2m} \to \mathbb{R}^q$. Then by setting

$$\begin{aligned} u_1(k) &= y_2(k) \\ u_2(k) &= y_1(k) \end{aligned} \tag{7-7}$$

The two sub-systems are connected and turned into a single closed-loop system.

More generally, consider an MMPS sub-system i with n connections to other sub-systems, then this system can be described by:

$$\begin{aligned} x_i(k) &= A_i \otimes B_i \left(\otimes' \left(C_i \cdot x_i(k-1) + D_i \cdot x_i(k) + E_i \cdot u(k) + \sum_{j=1}^{n} E_{ij}\, u_j(k) \right) \right) \\ y_i(k) &= F_i \otimes (H_i \otimes' (K_i \cdot x_i(k-1) + L_i \cdot x(k))) \end{aligned} \tag{7-8}$$

Where $u(k)$ is a regular input to the sub-system, $u_j(k)$ is the input from the j^{th} sub-system. $x_i(k)$ is the state of the sub-system and $y_i(k)$ is the output. Given the implicit nature of an MMPS sub-system, properties for monotonicity and non-expansiveness have not been derived in known literature yet. Thus, they will not be discussed here. However, the condition for time-invariance can be determined. [17] has derived the time-invariant conditions for implicit MMPS systems with inputs. Now, not only should partial-additive homogeneity hold for all temporal and quantity signals as well as the input, but also for all signals from other sub-systems. This expands the current time-invariance condition. The conditions for time-invariance of an MMPS sub-system are defined by:

Theorem 7.1. *(Time invariance of an implicit MMPS sub-system)*
An implicit MMPS sub-system described by (7-8) *in the ABCDE form is time invariant when the following properties hold*

$$\begin{aligned}\sum_{j\in\overline{n_\mathrm{t}}}\Big[\begin{array}{cccc} C_{i,11} & D_{i,11} & E_{i,11} & [E_{i1,11}\dots E_{in,11}]\end{array}\Big]_{\ell j} &= 1, \forall \ell \in \overline{p_\mathrm{t}}\\ \sum_{j\in\overline{n_\mathrm{q}}}\Big[\begin{array}{cccc} C_{i,21} & D_{i,21} & E_{i,21} & [E_{i1,21}\dots E_{in,21}]\end{array}\Big]_{tj} &= 0, \forall t \in \overline{p_\mathrm{q}}\end{aligned} \tag{7-9}$$

Proof. This is an extension of the time-invariant condition proposed in [17], by logically adding the sub-system connection channels. □

Another property that must be investigated is the property of solvability. From Chapter 3, recall that an MMPS system is solvable if there exists a matrix $T \in \mathbb{R}^{n\times n}$ such that $F = T \cdot S_A \cdot S_B \cdot S_D \cdot T^{-1}$ is a lower triangular matrix. Where

$$\begin{aligned}&[S_A]_{i,j} = \begin{cases} 1 \text{ if } [A]_{i,j} \neq \varepsilon \\ 0 \text{ if } [A]_{i,j} = \varepsilon \end{cases} \quad [S_B]_{i,j} = \begin{cases} 1 \text{ if } [B]_{i,j} \neq \top \\ 0 \text{ if } [B]_{i,j} = \top \end{cases}\\ &[S_D]_{i,j} = \begin{cases} 1 \text{ if } [D]_{i,j} \neq 0 \\ 0 \text{ if } [D]_{i,j} = 0 \end{cases}\end{aligned} \tag{7-10}$$

This condition must also hold for any sub-system. However, connecting solvable sub-systems can result in an unsolvable system. To understand this, an ABCDE form for connected sub-systems must be introduced.

Proposition 3. *(Open loop integrated sub-system model)*
The set of independent sub-systems can be made into an open loop integrated sub-system model described as follows;

$$\begin{bmatrix} x(k) \\ y(k) \end{bmatrix} = \underbrace{\begin{bmatrix} A & \epsilon \\ \epsilon & F \end{bmatrix}}_{\bar{A}} \otimes \left(\underbrace{\begin{bmatrix} B & \top \\ \top & H \end{bmatrix}}_{\bar{B}} \otimes' \left(\underbrace{\begin{bmatrix} C & 0 \\ K & 0 \end{bmatrix}}_{\bar{C}} \begin{bmatrix} x(k-1) \\ y(k-1) \end{bmatrix} + \underbrace{\begin{bmatrix} D & E \\ L & 0 \end{bmatrix}}_{\bar{D}} \begin{bmatrix} x(k) \\ y(k) \end{bmatrix} + \underbrace{\begin{bmatrix} E_u \\ 0 \end{bmatrix}}_{\bar{E}} \begin{bmatrix} u(k) \end{bmatrix} \right) \right) \tag{7-11}$$

Where $x(k)$ and $y(k)$ contains all $x_i(k)$'s and $y_i(k)$;s appended respectively, A, B, C, D, F, H, K, L are a block diagonal matrices of all A_i, B_i, C_i, D_i, F_i, H_i, K_i and L_i respectively.

E connects the correct y_i with the correct u_i, so the shape depends on the configuration of the specific use case.

Proof. This is an extension of the ABCDE canonical from proposed in [11] by logically appending the states and matrices. □

Whether or not a system is solvable boils down to whether an explicit mapping of the form (7-12) exists or not.

$$x(k) = f(x(k), x(k-1), u(k)) \Rightarrow x(k) = g(x(k-1), u(k)) \tag{7-12}$$

The external input $u(k)$, in an open-loop system, does not depend on the state $x(k)$, so it does not matter if it is a controlled system or an autonomous system when looking at the notion of solvability. In the case of a system consisting of a set of sub-systems, if all sub-systems are solvable, connecting them into a single system can result in an unsolvable system. The K and L matrices essentially determine where a cycle update happens, as they determine which components of the current state are used for the output of a given sub-system. When this output $y_i(k)$ gets put back into the system via the E matrix, a circuit can be created, which means one has an unsolvable system. In the case of a transportation system, this happens when a vehicle route does not get a cycle update at one of the nodes.

One can prevent this by designing the outputs of the sub-systems such that after one full round, a cycle update occurs. For transportation networks, it is easy to see when this should happen. This must happen when a vehicle is back at its starting location. In general, this will mean that for a given output pair (y_{ij}, y_{ji}), one outputs the current state and one the previous state. So $L_{ij} = 0$ while $L_{ji} \neq 0$. While this is not necessarily a requirement, it is a good rule of thumb. For example, for pass-through nodes/ subsystems, this is not the case, as the same vehicle must continue on in the same cycle. When there are n sub-systems, then there are at minimum n output vectors one must connect and at most $n \cdot (n-1)$ if every sub-system is connected to every other sub-system.

In most practical applications, y_{ij} will simply be a state vector, either of current states, previous states or a combination of the two. This has as a result that F and H are the max-plus and min-plus identity matrices, respectively. Writing a system in terms of sub-systems will result in an inflated system description since all outputs $y(k)$ are explicitly taken as a state. If one is to derive a description of a system from scratch, all these states can be internalised into the normal state description of $x(k)$. However, this does not necessarily mean that there are extra modes introduced, which would slow down any analysis. It does however, mean modelling larger systems will become easier.

In Proposition 3, only an open loop system was taken. However, in control theory, closed loop systems are equally, if not more important. Suppose the system of (7-11) is turned into a closed-loop system with a reference signal. Then this new system is given by:

Proposition 4. *(Integrated closed-loop sub-system model with reference)*
Combining the closed loop control signal for MMPS systems with the integrated sub-system model to the ABCDR form allows for a closed loop system description of a sub-system model

as follows;

$$\begin{bmatrix} x(k) \\ y(k) \\ u(k) \end{bmatrix} = \underbrace{\begin{bmatrix} A & \varepsilon & \varepsilon \\ \varepsilon & F_y & \varepsilon \\ \varepsilon & \varepsilon & F_u \end{bmatrix}}_{\overline{A}} \otimes \left(\underbrace{\begin{bmatrix} B & \top & \top \\ \top & H_y & \top \\ \top & \top & H_u \end{bmatrix}}_{\overline{B}} \otimes' \left(\underbrace{\begin{bmatrix} C & 0 & 0 \\ K & 0 & 0 \\ K_0 & 0 & L_0 \end{bmatrix}}_{\overline{C}} \cdot \begin{bmatrix} x(k-1) \\ y(k-1) \\ u(k-1) \end{bmatrix} \right.\right.$$
$$\left.\left. + \underbrace{\begin{bmatrix} D & E_y & E_u \\ L & 0 & 0 \\ K_1 & 0 & L_1 \end{bmatrix}}_{\overline{D}} \cdot \begin{bmatrix} x(k) \\ y(k) \\ u(k) \end{bmatrix} + \underbrace{\begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \\ R_{21} & R_{22} \end{bmatrix}}_{\overline{R}} \cdot r(k) \right)\right) \tag{7-13}$$

Proof. This is an extension of the open loop integrated sub-system model proposed in Proposition 3 by logically adding the closed loop controller and rearranging the system matrices. □

Similar to the open-loop integrated sub-system model, a time-invariance condition can also be derived for the closed-loop case. In contrast to the open loop system, now the states for the input $u(k)$ must also adhere to the time invariance properties, as well as the reference input $r(k)$ must also be included. The condition for a time invariant integrated closed loop implicit MMPS sub-system model with reference can be found in Theorem 7.2

Theorem 7.2. *(Time invariance of a integrated closed loop implicit MMPS sub-system model with reference)*
A closed loop implicit MMPS sub-system described by (7-13) *in the ABCDR form is time invariant when the following properties hold:*

$$\begin{aligned} \sum_{i \in \bar{n}_t + \bar{u}_t} \begin{bmatrix} C_{11} & 0 & 0 & D_{11} & E_{y,11} & E_{u,11} & R_{11,11} & R_{12,11} \\ k_{11} & 0 & 0 & L_{11} & 0 & 0 & 0 & 0 \\ K_{0,11} & 0 & L_{0,11} & K_{1,11} & 0 & L_{1,11} & R_{21,11} & R_{22,11} \end{bmatrix}_{t,\ell i} &= 1, \forall \ell \in \bar{p}_t \\ \sum_{i \in \bar{n}_t + \bar{u}_t} \begin{bmatrix} C_{21} & 0 & 0 & D_{21} & E_{y,21} & E_{u,21} & R_{11,21} & R_{12,21} \\ k_{21} & 0 & 0 & L_{21} & 0 & 0 & 0 & 0 \\ K_{0,21} & 0 & L_{0,21} & K_{1,21} & 0 & L_{1,21} & R_{21,21} & R_{22,21} \end{bmatrix}_{q,\ell i} &= 0, \forall \ell \in \bar{p}_t \end{aligned} \tag{7-14}$$

Proof. This is an extension of the time-invariant condition for closed loop MMPS systems proposed by [17]. By logically adding the sub-system connection channels. □

Since in the closed-loop case the input $u(k)$ is a function of both the state $x(k)$ and itself $u(k)$, solvability is not always guaranteed, given of course, that the open-loop system is solvable. The input $u(k)$ can be seen an extra state or set of states, and so, in closing the loop, solvability must be verified again. Since the solvability conditions only pertain to $\overline{A}$, $\overline{B}$ and $\overline{D}$ using the theory from Subsection 3-2-2, it can easily be verified whether the closed loop system is solvable. Also notice that $\overline{R}$ can never violate the solvability.

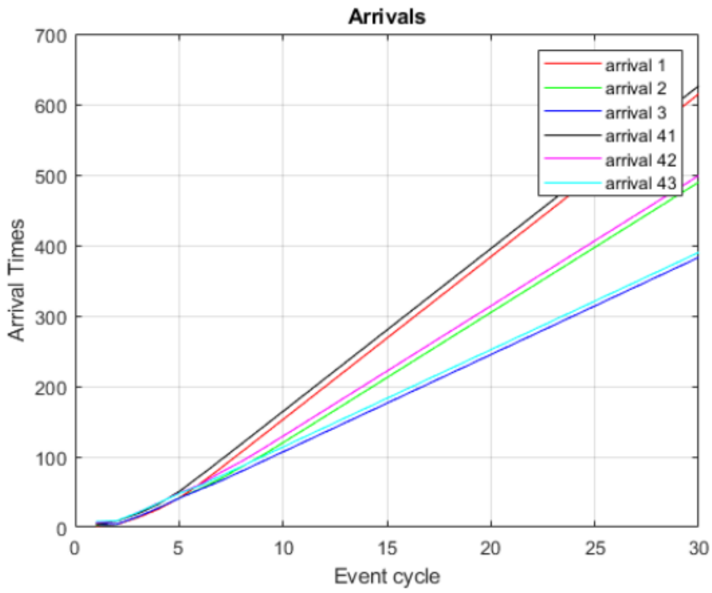


Figure 7-5: Arrival times of an unstable system [18]

7-1-3 Risks of Asynchronicity in Transportation Systems

A key property to note is the property of asynchronicity. Depending on the designed use case, one might be tempted to model a transportation system using an integrated sub-system model without forcing synchronisation between different vehicles or nodes. This section will briefly discuss why this practice should generally be avoided. In [18], some interesting results are obtained when simulating a 4-node transportation network with a bidirectional flow of goods. The premise was to remove the synchronisation requirement. This led to a stabilisation in quantity states but also to non-uniform state growth within a cycle for the time states. These two behaviours seem to contradict each other, as non-uniform state growth within a cycle indicates an unstable system, whilst stabilising quantity states indicate stable behaviour. This system, however, was not stable; removing synchronisation will always lead to an unstable system, unless the growth rate of the disconnected circuits are identical. This can be seen when one looks at the physical interpretation of the system's quantity states. Taking a look at Figure 7-5, one sees the arrival times of the unstable system from [18]; the departure times look very similar. Notice that *arrival 43* is modelled as processing the goods from *arrival 41* in, for example, cycle 15. Since the state equations only consider the cycle index and not the actual arrival times, the model assumes the goods from *arrival 41* are already present in that cycle. In reality, however, these goods arrive about 100 time units later, by which point *arrival 43* has already advanced by roughly 8 cycles. So goods are processed at another plant while they have not arrived yet. This is inherent to systems without proper synchronisation. Once again, this is because the state evolution is based on cycles and not actual times. The system therefore, processes goods that either are not there or believe goods have not yet arrived but already have. This poses a significant challenge that must be addressed carefully in the design process.

If one wishes to have a non-synchronous system. This is possible only for a short while,

while actively detecting when the system is no longer valid. In this scenario, one can model the system as a switching MMPS system such that when a buffer is empty or overflowing, it switches to another synchronised system, or one that can upscale to process the excess. Alternatively, one can consider an alternative modelling method, one that is better suited for unsynchronised systems.

7-2 Switching MMPS Systems as a Single System

Switching Max-Min-Plus-Scaling (S-MMPS) systems are discrete event systems that can switch between various modes of operation, where every mode is described by its own MMPS system. This mode can be determined based on the previous states, the current state, the previous mode or even by some external signal. It can also happen that this switching signal is arbitrary and can not be determined in advance. Modelling systems as S-MMPS systems can be very useful in flexible systems such as flexible production systems, traffic light intersections or transportation systems. S-MMPS systems are defined as follows:

Definition 7.1. *(Switching MMPS systems [19])*
A Switching MMPS system can be described by:

$$x(k) = A(\ell(k)) \otimes (B(\ell(k)) \otimes' (C(\ell(k)) \cdot x(k-1) + D(\ell(k)) \cdot x(k))) \tag{7-15}$$

Where the matrices $A(\ell) \in \mathbb{R}_{\varepsilon}^{n\times m}, B(\ell) \in \mathbb{R}_{\top}^{m\times p}$*,* $C(\ell(k)) \in \mathbb{R}^{p\times n}$*,* $D(\ell(k)) \in \mathbb{R}^{p\times n}$ *are the system matrices for the* ℓ*-th mode,* $\ell \in \{1, \ldots, n_L\}$*. and* n_L *is the number of modes the system has.*

Switching systems are hard to analyse due to their switching behaviour, even if the switching signal is known beforehand. Furthermore, two stable systems can still become unstable under the wrong switching rule and vice versa [20].

Suppose one has n_L different switching modes and the switching signal is known and can be written as an MMPS function. Then it is possible to write the S-MMPS system as an extended regular MMPS system as such:

Proposition 5. *(A S-MMPS system as as single MMPS system)*
Any Switching MMPS system with n_L *different switching modes, that has a switching signal* $\ell(k)$ *that can be represented as one or more MMPS functions, can be written as a single MMPS system.*

Proof. Introduce a state $c(k) \in \mathbb{R}^{n_L}$. Since $\ell(k)$ can be represented as an MMPS function, one can design an MMPS system such that $c_i = 0$ if mode i is active and $c_i << 0$ if mode i is not active. Which can be described by

$$c(k) = A_c \otimes B_c \otimes' (C_{cc} \cdot c(k-1) + D_{cc} \cdot c(k) + C_{cx} \cdot x(k-1) + D_{cx} \cdot x(k-1) + E \cdot u_c(k)) \tag{7-16}$$

$$\text{This means that for all } x(k), \text{ there exist exactly one } \; i \in \{1, \ldots, n_L\}$$
$$\text{such that } c_i = 0 \text{ and } c_j \ll 0 \quad \forall j \in \{1, \ldots, n_L\},\; j \neq i$$

Then

$$\begin{bmatrix} x(k) \\ c(k) \end{bmatrix} = \left[\begin{array}{ccc|c} A_1 & \dots & A_{n_L} & \mathcal{E} \\ \hline \mathcal{E} & \dots & \mathcal{E} & A_c \end{array}\right] \otimes \left(\left[\begin{array}{ccc|c} B_1 & & \top & \top \\ & \ddots & & \vdots \\ \top & & B_{n_L} & \top \\ \hline \top & \dots & \top & B_c \end{array}\right] \otimes' \right.$$

$$\left.\left(\left[\begin{array}{c|c} C_1 & 0 \\ \vdots & \vdots \\ C_{n_L} & 0 \\ \hline C_{cx} & C_{cc} \end{array}\right] \cdot \begin{bmatrix} x(k-1) \\ c(k-1) \end{bmatrix} + \left[\begin{array}{c|c} D_1 & D_{c1} \\ \vdots & \vdots \\ D_{n_L} & D_{cn_L} \\ \hline D_{cx} & D_{cc} \end{array}\right] \cdot \begin{bmatrix} x(k) \\ c(k) \end{bmatrix} + \left[\begin{array}{c|c} E_1 & 0 \\ \vdots & \vdots \\ E_{n_L} & 0 \\ \hline E_{cx} & E_{cc} \end{array}\right] \cdot \begin{bmatrix} u(k) \\ u_c(k) \end{bmatrix} \right)\right) \tag{7-17}$$

Is the S-MMPS system reformulated into a regular MMPS system. Where $A_i \in \mathbb{R}_{\varepsilon}^{n\times m}$ $\forall i \neq c$, $A_c \in \mathbb{R}_{\varepsilon}^{n_L\times q}$, $B_i \in \mathbb{R}_{\top}^{m\times l}$ $\forall i \neq c$, $B_c \in \mathbb{R}_{\top}^{q\times k}$, $C_i, D_i \in \mathbb{R}^{l\times n}$ $\forall i = \{1,\dots,n_L\}$,$D_{ci} \in \mathbb{R}^{l\times n_L}$, $C_{cc}, D_{cc} \in \mathbb{R}^{k\times n_L}$, $C_{cx}, D_{cx} \in \mathbb{R}^{k\times n}$, $E_i \in \mathbb{R}^{l\times n_u}$ $\forall i \neq cx, cc$, $E_{cx} \in \mathbb{R}^{k\times n_u}$, $E_{cc} \in \mathbb{R}^{k\times n_L}$

And the matrix $D_{ci} \in \mathbb{R}^{m\times n}$ has all entries zero except for the i^{th} column, which is filled with a large constant M.

If the switching signal can be represented as an MMPS function, then one can construct the mode variables c_i such that they are 0 when the system is in mode i and < 0 if the system is not in mode i.
Now suppose that mode i is active. $c_i = 0$ and $c_j < 0 \quad \forall j \in \{1,\dots,n_L\},\ j \neq i$. Take $D_{ck} \in \mathbb{R}^{p\times n_L}$ as a matrix filled with zeros and M in the k^{th} column as such:

$$D_{ck} = \begin{array}{c} \overbrace{\hphantom{0 \quad \cdots \quad 0 \quad M}}^{k^{th} column} \\ \begin{bmatrix} 0 & \cdots & 0 & M & 0 & \cdots & 0 \\ 0 & \cdots & 0 & M & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & M & 0 & \cdots & 0 \end{bmatrix} \end{array} \tag{7-18}$$

Then $D_{ci} \cdot c_i = \mathbf{0}$ while $D_{cj} \cdot c_j <<< \mathbf{0}$. Since this large negative vector will be added to $C_j \cdot x(k-1) + D_j \cdot x(k)$, it means that (7-19) is true if M is large enough.

$$A_j \otimes B_j \otimes' C_j \cdot x(k-1) + D_j \cdot x(k) + D_{cj} \cdot c(k) < A_i \otimes B_i \otimes' C_i \cdot x(k-1) + D_i \cdot x(k) + D_{ci} \cdot c(k) \tag{7-19}$$

Even though time states are continuous. There is always a finite difference between the state of the active mode and the inactive mode. Since there is a difference, there also must exist an M large enough to make (7-19) true for all i, j, k. Also, when one or more of the switching modes have a growth rate of zero, this method still works because it does not explicitly rely on the growth rate. The switching is based on the switching state c, which determines the state independently of any growth rate of a mode.

□

It still remains unclear how large M should be, as this depends on the system and the starting position. This method is useful, for example for systems with different operating modes where the system is bound to different rules, interactions or dynamics in the different modes. Below one will find a small example of an S-MMPS system turned into a single MMPS system.

Example 7.1. *(Transforming a SMMPS system into a single MMPS system)*
Consider a switching MMPS system with 2 modes
Where mode 1 is given by:

$$x(k) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \left(\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} \otimes' \left(\begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} x(k-1) + \begin{bmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{bmatrix} x(k) \right) \right) \tag{7-20}$$

And mode 2 is given by:

$$x(k) = \begin{bmatrix} 6 & 7 \\ 8 & 9 \end{bmatrix} \otimes \left(\begin{bmatrix} -1 & 2 \\ 5 & 4 \end{bmatrix} \otimes' \left(\begin{bmatrix} 0 & \frac{3}{4} \\ 1 & 0 \end{bmatrix} x(k-1) + \begin{bmatrix} 0 & \frac{1}{4} \\ 0 & 0 \end{bmatrix} x(k) \right) \right) \tag{7-21}$$

The system has a switching input which is given by $u_c(k)$, which is one when the system is in mode 1 and two when the system is in mode 2.

Then $c_1(k)$ and $c_2(k)$ are given by:

$$\begin{aligned} c_1(x) &= \min\left(0,\, 1 - u_c(k), u_c(k) - 1\right), \\ c_2(x) &= \min\left(0,\, 2 - u_c(k), u_c(k) - 2\right). \end{aligned} \tag{7-22}$$

Then, using Proposition 5, the S-MMPS system can be written as a single MMPS system as such:

$$\begin{bmatrix} x(k) \\ c(k) \end{bmatrix} = \begin{bmatrix} 1 & 2 & \epsilon & \epsilon & \epsilon & \epsilon \\ 3 & 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 6 & 7 & \epsilon & \epsilon \\ \epsilon & \epsilon & 8 & 9 & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 \end{bmatrix} \otimes \left(\begin{bmatrix} 2 & 1 & \top & \top & \top & \top & \top & \top & \top \\ 4 & 2 & \top & \top & \top & \top & \top & \top & \top \\ \top & \top & -1 & 2 & \top & \top & \top & \top & \top \\ \top & \top & 5 & 4 & \top & \top & \top & \top & \top \\ \top & \top & \top & \top & 1 & -1 & 0 & \top & \top \\ \top & \top & \top & \top & \top & \top & 0 & 2 & -2 \end{bmatrix} \otimes' \right.$$
$$\left. \left(\begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x(k-1) \\ c(k-1) \end{bmatrix} + \begin{bmatrix} 0 & \frac{1}{2} & M & 0 \\ 0 & 0 & M & 0 \\ 0 & \frac{1}{4} & 0 & M \\ 0 & 0 & 0 & M \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x(k) \\ c(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 2 \\ -2 \end{bmatrix} \cdot \begin{bmatrix} u_c(k) \end{bmatrix} \right) \right) \tag{7-23}$$

Where M is a sufficiently large number.

7-3 Transportation Network Framework

This section describes several building blocks one can use for easy construction of a transportation network. A transportation network is a very general concept which can refer to any kind of logistical system. Thus, the hereafter-referred vehicles can, depending on the real-world equivalent, be any type of vehicle from trucks to ships, from drones to trains. This section begins with describing a central node, which can be considered the basis node from which a system can be built. This central node has 3 arms connecting to any other node, allowing for complex modelling. The second part of this section highlights 4 other types of nodes which can be connected to such a central node or even to each other in some cases. Lastly, other, more complex nodes are briefly introduced, which allow for even more detailed modelling.

Since nodes have been 'cut loose' from their network, it might become unclear which part of the state equations belongs to the node dynamics and which are simply inputs to the system. As the inputs are referred to by their physical interpretation, such as a departure time instead of an input. To quickly see what states belong to the input of the node, it is decided to make these states blue. For example, take the arrival time at node 1 as the departure time of a specific vehicle at node 2 plus travel time τ. This will then be denoted by

$$a_1(k) = d_2(k) + \tau \tag{7-24}$$

Here $d_2(k)$ clearly is the departure time from another node, and thus will be considered as an input to the node; hence it has been made blue.

Several variables are used during the modelling, instead of introducing them every time they are listed in Table 7-1

Variable	Definition
τ_{ij}	Travel time from node i to node j
$\rho_{\max,i}$	Maximum capacity of vehicle i
u_i	Unloading speed of vehicle i
φ	Outflow of goods per unit of time
L_i	Loading speed of vehicle i
β_i, $(1-\beta_i)$	Fraction of the load of vehicle i going to another vehicle
γ	Inflow of goods per unit of time

Table 7-1: Common variable definition

Since the modelling happens in a DE framework, it is important to note what the different elements refer to. All the states represent the time at which a specific event occurs for the k^{th} time. Take, for example $a_{1,1}(2) = 5$, where $a_{1,1}$ is the arrival time of vehicle 1 at node 1. $a_{1,1}(2) = 5$ means that vehicle 1 has arrived for the second time at node 1 at time 5. Which could refer to any time quantity, hours, seconds, this all depends on the definition of the designer. This means that k is the cycle counter of the system. Quantity states represent a quantity at a specific event in a specific cycle and are always connected to a time state. So if s is a quantity state referring to the number of parcels waiting to be picked up and is linked to some arrival time, $s(4) = 10$ will mean that in cycle 4, 10 parcels are waiting to be picked up at the moment the arrival happens.

A few last remarks about how the different nodes are modelled. The first subscript of a state says what node it is referring to, and the second is from where it came or in other words, refers to a specific vehicle. All nodes have been checked and are time-invariant. It has been assumed that for all vehicles, the unloading speed is higher than the loading speed. This makes modelling easier as the restricting speed will always be the loading speed.

7-3-1 Basic Central Node Structure

This subsection describes the working of a central node without a stack. The central node has 3 arms which extend out to a connection node. It is not necessary to know what type of connection node is present for the working of the central node; they are, in a sense, independent of each other. A visual representation can be found in Figure 7-6. Note that node 4 is the central node and nodes 1, 2 and 3 are only present as a visual aid. The interactions of several connection nodes are described in Subsection 7-3-2.

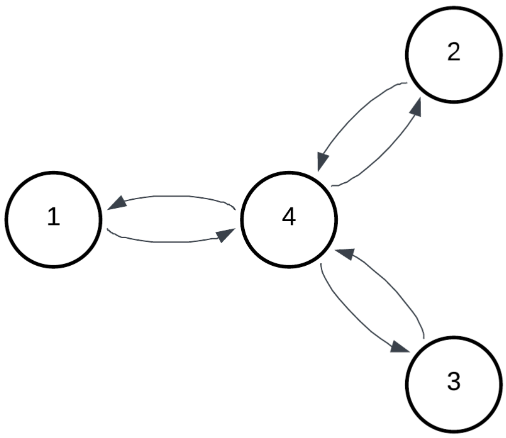


Figure 7-6: Visual representation of a 3-armed central node

In the case where there is no storage stack present at the central node, it is required that all goods that come in in cycle k also leave the node in the same cycle k. Goods can flow in all directions, so goods coming from node 1 can go to node 2 and 3, goods coming from node 2 can go to node 1 and 3 and goods coming from node 3 can go to node 1 and 2. A fixed fraction of the incoming goods at each node is always routed to specific downstream nodes. For example, the goods arriving from node 1, denoted by $\rho_1(k)$, are consistently split between nodes 2 and 3. A constant fraction $\beta_1 \in [0,1]$ is directed to node 2, meaning that $\beta_1\rho_1(k)$ goods are sent to node 2, while the remaining $(1-\beta_1)\rho_1(k)$ goods are sent to node 3. β_2 and β_3 naturally denote the fractional splits between the goods arriving from node 2 and node 3 respectively. The arrival of vehicles 1, 2 and 3 at node 4, denoted by $a_{41}(k)$, $a_{42}(k)$, $a_{43}(k)$ in cycle k respectively, is described as follows:

$$\begin{aligned} a_{41}(k) &= d_1(k) + \tau_{14} \\ a_{42}(k) &= d_2(k) + \tau_{24} \\ a_{43}(k) &= d_3(k) + \tau_{34} \end{aligned} \tag{7-25}$$

Where $d_i(k)$ denotes the departure time of the vehicle departing from node i.

The times when a vehicle's 1, 2 and 3 are empty at node 4 in cycle k are given by $e_{41}(k)$, $e_{42}(k)$, $e_{43}(k)$ respectively. Which can be described as follows:

$$
\begin{aligned}
e_{41}(k) &= a_{41}(k) + \frac{\rho_1(k)}{u_1} \\
e_{42}(k) &= a_{42}(k) + \frac{\rho_2(k)}{u_2} \\
e_{43}(k) &= a_{43}(k) + \frac{\rho_3(k)}{u_3}
\end{aligned}
\tag{7-26}
$$

Where ρ_i is the load of vehicle i when arriving at node 4.

The departure times are a bit more complex. Only the departure time of vehicle 1 will be explained; however, for the other vehicles the same holds. It is assumed that vehicle 1 can start loading the goods from vehicle 2 only when both vehicles 1 and 2 are empty. The same holds for vehicle 1 and vehicle 3. Then vehicle 1 can leave when the last truck is emptied, plus the time it takes to load the goods destined for vehicle 1. This leads to the following departure time:

$$
d_{41}(k) = \max\left(\max(e_{41}(k), e_{42}(k)) + \frac{\beta_2\rho_2(k)}{L_1}, \max(e_{41}(k), e_{43}(k)) + \frac{\beta_3\rho_3(k)}{L_1}\right) \tag{7-27}
$$

However, it can also happen that the empty times are so close together that one pair is still loading while the other vehicle has already emptied itself. In other words, vehicle 1 is still loading the final goods from vehicle 2, but vehicle 3 has already finished emptying. In this case, vehicle 1 can leave at the moment loading started plus the time it takes to load all the goods from vehicle 2, plus the time it takes to load all the goods from vehicle 3. The moment loading started is denoted by δ_1:

$$
\delta_1(k) = \min\left(\max(e_{41}(k), e_{42}(k)), \max(e_{41}(k), e_{43}(k))\right) \tag{7-28}
$$

And the time it takes to load all goods destined for vehicle 1 is given by:

$$
\frac{\beta_2\rho_2(k) + \beta_3\rho_3(k)}{L_1} \tag{7-29}
$$

Resulting in the final departure time $d_{41}(k)$:

$$
\begin{aligned}
d_{41}(k) = \max\Big(&e_{41}(k) + \frac{\beta_2\rho_2(k)}{L_1}, e_{42}(k) + \frac{\beta_2\rho_2(k)}{L_1}, e_{41}(k) + \frac{\beta_3\rho_3(k)}{L_1}, e_{43}(k) + \frac{\beta_3\rho_3(k)}{L_1}, \\
&\delta_1 + \frac{\beta_2\rho_2(k) + \beta_3\rho_3(k)}{L_1}\Big)
\end{aligned}
\tag{7-30}
$$

The departure times of vehicles 2 and 3 are very similar and can be found below:

$$
\begin{aligned}
d_{42}(k) &= \max\Big(e_{42}(k) + \frac{\beta_1\rho_1(k)}{L_2}, e_{41}(k) + \frac{\beta_1\rho_1(k)}{L_2}, e_{42}(k) + \frac{(1-\beta_3)\rho_3(k)}{L_2},\\
&\qquad e_{43}(k) + \frac{(1-\beta_3)\rho_3(k)L_3}{,}\delta_2 + \frac{\beta_1\rho_1(k) + (1-\beta_3)\rho_3(k)}{L_2}\Big)\\
d_{43}(k) &= \max\Big(e_{43}(k) + \frac{(1-\beta_1)\rho_1(k)}{L_3}, e_{41}(k) + \frac{(1-\beta_1)\rho_1(k)}{L_3}, e_{43}(k) + \frac{(1-\beta_2)\rho_2(k)}{L_3},\\
&\qquad e_{42}(k) + \frac{(1-\beta_2)\rho_2(k)}{L_3}, \delta_3 + \frac{(1-\beta_1)\rho_1(k) + (1-\beta_2)\rho_2(k)}{L_3}\Big)\\
\delta_2(k) &= \min\big(\max(e_{42}(k), e_{41}(k)), \max(e_{42}(k), e_{43}(k))\big)\\
\delta_3(k) &= \min\big(\max(e_{43}(k), e_{41}(k)), \max(e_{43}(k), e_{42}(k))\big)
\end{aligned}
\tag{7-31}
$$

Lastly, the vehicle loads at departure are needed. These can also easily be derived since they are the fractional loads assigned to them from the arriving vehicles:

$$
\begin{aligned}
\rho_{41}(k) &= \beta_2\rho_2(k) + \beta_3\rho_3(k)\\
\rho_{42}(k) &= \beta_1\rho_1(k) + (1-\beta_3)\rho_3(k)\\
\rho_{43}(k) &= (1-\beta_1)\rho_1(k) + (1-\beta_2)\rho_2(k)
\end{aligned}
\tag{7-32}
$$

7-3-2 Basic Connection Node Types

There are numerous types of nodes one can design by tweaking the iterations one desires. In this section, four different connection nodes are described. They are an input node, an output node, a transfer node without stack and a pass-through node. Several other types of nodes and their equations of state can be found in Appendix A. The nodes are all referred to as node n and are connected to either one or two nodes. Then the one or two refers to whichever node or input is taken.

Input node

An input node is a node where a vehicle will arrive empty and only pick up goods. This, for example, can be a producer of goods, such as a farmer or a manufacturing plant. Such a node has an input at a rate of γ goods per unit of time. These goods get stored in a stack s. When a vehicle arrives, it will transfer goods from the stack to its internal storage ρ; when the stack is empty or the vehicle is full, it will leave again. In Figure 7-7, a visual representation can be seen of such a node.

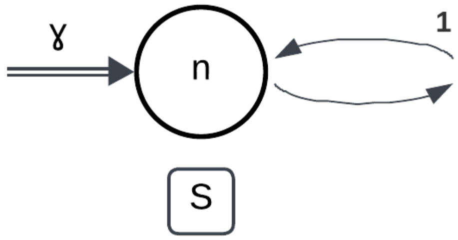


Figure 7-7: Visual representation of an input Node

Firstly, the function to determine the arrival time is modelled. This is quite easy as the arrival time of vehicle 1 at node n is just the departure time of vehicle 1 at the previous node plus the travel time τ_{1n}. resulting in:

$$a_{n1}(k) = d_1(k) + \tau_{1n} \tag{7-33}$$

The size of the stack $s_n(k)$ at departure can be determined by taking the stack of the previous cycle $s_n(k-1)$ and looking at what was added and taken away in that cycle. $d_{n1}(k)$ denotes the departure of vehicle 1 from node n. So what was added as a result of the constant input is given by:

$$\gamma(d_{n1}(k) - d_{n1}(k-1)) \tag{7-34}$$

This is the time difference between the departure of the vehicle in the previous cycle and the current cycle, multiplied by the input rate γ. To determine what was taken, the difference between the arrival and departure of the current cycle is taken and multiplied by the loading rate L_1.

$$L_1(d_{n1}(k) - a_{n1}(k)) \tag{7-35}$$

Combining the two yields the function for the stack as:

$$s_n(k) = s_n(k-1) + \gamma(d_{n1}(k) - d_{n1}(k-1)) - L_1(d_{n1}(k) - a_{n1}(k)) \tag{7-36}$$

A vehicle will depart either when it is full or when the stack is empty. First, let's look at when a vehicle leaves because it is full. Since the vehicle arriving will be empty, one can take the arrival time and simply add the time it takes to fill up the vehicle. This time can be obtained by dividing the capacity of the vehicle $\rho_{\max,1}$ by the loading speed L_1, $\frac{\rho_{\max,1}}{L_1}$. Thus, the departure of a full truck in cycle k is given by:

$$d_{full}(k) = a_{n1}(k) + \frac{\rho_{\max,1}}{L_1} \tag{7-37}$$

To determine the departure when the stack is empty, one must consider the amount of goods that one wants to load and the amount of goods that can be loaded in that time. The amount of goods one loads can be found by taking the stop-time, i.e. $d_{n1}(k) - a_{n1}(k)$, and multiplying it by the loading speed:

$$\text{goods that where loaded} = L_1(d_{n1}(k) - a_{n1}(k)) \tag{7-38}$$

Then the goods that want to be loaded are goods that were left on the stack the previous cycle, so $s_n(k-1)$ and the goods that were added due to the input. Which is given by $\gamma(d_{n1}(k) - d_{n1}(k-1))$.

$$\text{goods that want to load} = s(k-1) + \gamma(d_{n1}(k) - d_{n1}(k-1)) \tag{7-39}$$

For the stack to be emptied, the goods that can be loaded and the goods that are loaded must be equal, so:

$$L_1(d_{n1}(k) - a_{n1}(k)) = s(k-1) + \gamma(d_{n1}(k) - d_{n1}(k-1)) \tag{7-40}$$

Here we are interested in finding the departure time $d_{n1}(k)$, so by performing some algebraic operations, one can find the departure time, which is given as follows:

$$d_{n1}(k) = (L_1 - \gamma)^{-1} \cdot (s_n(k-1) + L_1 a_{n1}(k) - \gamma d_{n1}(k-1)) \tag{7-41}$$

Then, by combining the two found equations, one can obtain the final departure time. Since the stack can be larger than what a vehicle can take, one must make sure that if that happens, the vehicle will leave as soon as it is full. One can obtain this by taking the minimum of both of the presented departure times:

$$d_{n1}(k) = \min\left(a_{n1}(k) + \frac{\rho_{\max,1}}{L}, (L_1 - \gamma)^{-1} \cdot (s_n(k-1) + L_1 a_{n1}(k) - \gamma d_{n1}(k-1))\right) \tag{7-42}$$

Lastly, the load of the vehicle $\rho_{n1}(k)$ must be determined. This was already presented in (7-38) and is the difference between the departure and arrival times, multiplied by the load speed:

$$\rho_{n1}(k) = L_1(d_{n1}(k) - a_{n1}(k)) \tag{7-43}$$

This results in the final description of the input node, as can be found below:

$$\begin{aligned} a_{n1}(k) &= d_1(k) + \tau_{1n} \\ s_n(k) &= s_n(k-1) + \gamma(d_{n1}(k) - d_{n1}(k-1)) - L_1(d_{n1}(k) - a_{n1}(k)) \\ d_{n1}(k) &= \min\left(a_{n1}(k) + \frac{\rho_{\max,1}}{L}, (L_1 - \gamma)^{-1} \cdot (s_n(k-1) + L_1 a_{n1}(k) - \gamma d_{n1}(k-1))\right) \\ \rho_{n1}(k) &= L_1(d_{n1}(k) - a_{n1}(k)) \end{aligned} \tag{7-44}$$

It is essential to ensure that L_1 is significantly larger than γ. Otherwise, the node will be unstable. This is intuitive; if the inflow of goods exceeds the loading capacity of the vehicle, it becomes impossible for the vehicle to transport all incoming goods, leading to unbounded growth of the stack at the node.

Output node

An output node is very similar to an input node. However, in this case a vehicle will arrive with goods and will drop off everything, after which it leaves. This could, for example, be an end user like the last stop of a delivery driver or a supermarket that receives resupplies daily. Such an output node has an output of goods at a rate of φ goods per unit of time. Again, these goods are stored in the stack s until they are delivered. When a vehicle arrives, it starts unloading and transfers all its goods to the stack. The moment the vehicle is empty, it will leave again. In Figure 7-8, a visual representation can be seen of such a node.

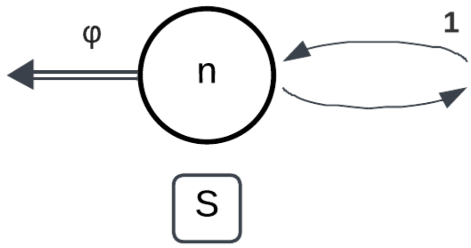


Figure 7-8: Visual representation of an output node

The arrival time for the output node $a_{n1}(k)$ is the same as for the input node. It is equal to the departure of vehicle 1 at the node of origin $d_1(k)$ with the travel time τ_{1n} added, resulting in:

$$a_{n1}(k) = d_1(k) + \tau_{1n} \tag{7-45}$$

The departure time of vehicle 1 at an output node is the moment the entire load has been unloaded. The time it takes to unload is given by the load divided by the unloading speed. Unloading can start as soon as vehicle 1 has arrived at the output node. Thus, the departure of vehicle 1 at an output node is given by:

$$d_{n1}(k) = a_{n1}(k) + \frac{\rho_1(k)}{u_1} \tag{7-46}$$

An output node also has a stack, as the inflow of goods from a vehicle might not be able to immediately leave the node. The size of the stack is modelled at the moment the vehicle leaves. The stack size $s_n(k)$ can be determined by taking the stack of the previous cycle $s_n(k-1)$ and looking at what was added and removed. The entire vehicle load $\rho_1(k)$. What was removed is given by:

$$\varphi(d_{n1}(k) - d_{n1}(k-1)) \tag{7-47}$$

This is the time difference between the departure of the vehicle in the previous cycle and the current cycle, multiplied by the output rate φ. This results in the following equation for the stack size:

$$s_n(k) = s_n(k-1) + \rho_1(k) - \varphi(d_{n1}(k) - d_{n1}(k-1)) \tag{7-48}$$

It is important to make sure the outflow of goods is much larger than the inflow of goods. Otherwise, the inflow can not be outputted, which will result in an increasing stack size, resulting in an unstable system. One must also consider that the stack can never become negative, since it is not possible to output goods that you do not have. Therefore, we must make sure the stack never becomes negative. This resulting in the following final stack equation:

$$s_n(k) = \max\left(s_n(k-1) + \rho_1(k) - \varphi(d_{n1}(k) - d_{n1}(k-1)), 0\right) \tag{7-49}$$

This results in the final description of the output node:

$$\begin{aligned} a_{n1}(k) &= d_1(k) + \tau_{1n} \\ s_n(k) &= \max\left(s_n(k-1) + \rho_1(k) - \varphi(d_{n1}(k) - d_{n1}(k-1)), 0\right) \\ d_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \end{aligned} \tag{7-50}$$

Transfer node without stack

A transfer node without a stack is a node where all goods are transferred from one vehicle to another. This can be, for example two drivers exchanging trailers at a border, or goods going to another type of vehicle of similar size. Since this node is connected to two other nodes, node 1 and node 2, there are two vehicles involved. A visual representation of such a transfer scenario is shown in Figure 7-9.

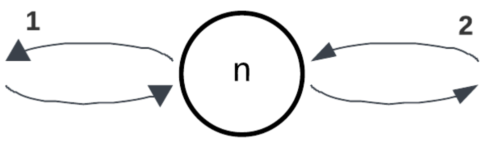


Figure 7-9: Visual representation of a transfer node without stack

The arrival time of vehicle 1 at node n $a_{n1}(k)$ is given by the departure time at node 1 plus the travel time τ_{1n}. For vehicle 2, the same holds with respect to node 2. Resulting in arrival times:

$$\begin{aligned} a_{n1}(k) &= d_1(k) + \tau_{1n} \\ a_{n2}(k) &= d_2(k) + \tau_{2n} \end{aligned} \tag{7-51}$$

The time when a vehicle is empty is given by the time it takes to unload all the goods plus the arrival time, which for both vehicles is given by $e_{n1}(k)$ and $e_{n2}(k)$:

$$\begin{aligned} e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \\ e_{n2}(k) &= a_{n2}(k) + \frac{\rho_2(k)}{u_2} \end{aligned} \tag{7-52}$$

A vehicle can leave once all the goods for that vehicle are loaded. Which is the total load of the other vehicle. The time it takes to load these goods is given by dividing the load by the loading speed. Loading can only start when both vehicles are empty. Once all goods are loaded, the vehicle will immediately leave. Thus the departure times of vehicle 1 and 2, $d_{n1}(k)$ and $d_{n2}(k)$ respectively, are given by:

$$\begin{aligned} d_{n1}(k) &= \max\left(e_{n1}(k), e_{n2}(k)\right) + \frac{\rho_2(k)}{L_1} \\ d_{n2}(k) &= \max\left(e_{n1}(k), e_{n2}(k)\right) + \frac{\rho_1(k)}{L_2} \end{aligned} \tag{7-53}$$

Finally, the loads of the departing vehicles, $\rho_{n1}(k)$ and $\rho_{n2}(k)$, correspond to the loads of the vehicles arriving from the opposite direction. In other words, each departing vehicle continues with the load brought in by the other arriving vehicle:

$$\begin{aligned} \rho_{n1}(k) &= \rho_2(k) \\ \rho_{n2}(k) &= \rho_1(k) \end{aligned} \tag{7-54}$$

This results in the final description for a transfer node without a stack:

$$\begin{aligned} a_{n1}(k) &= d_1(k) + \tau_{1n} \\ a_{n2}(k) &= d_2(k) + \tau_{2n} \\ e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \\ e_{n2}(k) &= a_{n2}(k) + \frac{\rho_2(k)}{u_2} \\ d_{n1}(k) &= \max\left(e_{n1}(k), e_{n2}(k)\right) + \frac{\rho_2(k)}{L_1} \\ d_{n2}(k) &= \max\left(e_{n1}(k), e_{n2}(k)\right) + \frac{\rho_1(k)}{L_2} \\ \rho_{n1}(k) &= \rho_2(k) \\ \rho_{n2}(k) &= \rho_1(k) \end{aligned} \tag{7-55}$$

Pass-through node

A pass-through node is a node where, depending on the design, one or two vehicles pass through without interacting and simply continue their journey. While this type of node may not have a direct practical application on its own, it can be useful for modelling purposes. For instance, it allows two central nodes to be combined into a single node with four connections instead of three. Additionally, pass-through nodes become relevant when inputs or outputs are added, such as at customer locations where goods are picked up or dropped off while the vehicle continues on its route. Such a node is visualised in Figure 7-10.

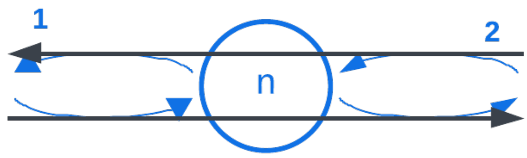


Figure 7-10: Visual representation of a Pass-through node

As mentioned earlier, depending on the use case, a pass-through node can either involve two vehicles passing through independently or a single vehicle travelling back and forth in a cycle. In this model, we consider the latter case; a single vehicle that travels between node 1 and node 2 through the pass-through node n in each cycle.

Here, $a_{n1}(k)$ denotes the arrival time at node n of the vehicle travelling from node 1 to node 2 during cycle k, while $a_{n2}(k)$ denotes the arrival time at node n for the return trip from node 2 to node 1 in the same cycle. Both values are determined by the departure time from the origin node and the corresponding travel time:

$$\begin{aligned} a_{n1}(k) &= d_1(k) + \tau_{1n} \\ a_{n2}(k) &= d_2(k) + \tau_{2n} \end{aligned} \tag{7-56}$$

In general, when using this basic pass-through node setup, it is preferable to define travel times directly between node 1 and node 2 and between node 2 and node 1. Splitting these times across the pass-through node can lead to confusion, since the node only serves as a modelling construct and not as a physical location. However, if additional actions take place at the pass-through node, such as picking up or dropping off goods, this simplification no longer applies because the node then represents a meaningful location in the network.

Since there are no actions performed at a pass-through node, the departure times are equal to the arrival times:

$$\begin{aligned} d_{n1}(k) &= a_{n2}(k) \\ d_{n2}(k) &= a_{n1}(k) \end{aligned} \tag{7-57}$$

Lastly, the loads of the vehicles are mathematically swapped. In reality, of course, the vehicle simply continues on its journey:

$$\begin{aligned} \rho_{n1}(k) &= \rho_2(k) \\ \rho_{n2}(k) &= \rho_1(k) \end{aligned} \tag{7-58}$$

This results in the complete description of a pass-through node:

$$\begin{aligned} a_{n1}(k) &= d_1(k) + \tau \\ a_{n2}(k) &= d_2(k) + \tau \\ d_{n1}(k) &= a_{n2}(k) \\ d_{n2}(k) &= a_{n1}(k) \\ \rho_{n1}(k) &= \rho_2(k) \\ \rho_{n2}(k) &= \rho_1(k) \end{aligned} \tag{7-59}$$

7-3-3 Advanced Node Extensions

In this section, potential extensions to the existing node framework are discussed that go beyond the basic node types introduced earlier. This includes considerations for more complex node configurations, such as nodes with more than three arms, as well as the challenges and methods involved in modelling multiple vehicles on the same route. These extensions open up new possibilities for designing more realistic and scalable transport systems, while highlighting some of the mathematical and modelling complexities that arise. In Appendix A, one can find a visual representation as well as the system of equations for the following node types:

- In- and output node
- Transfer node with stack
- Transfer node with input
- Transfer node with output
- Transfer node with input and output
- Pass-through node with input
- Pass-through node with output
- Pass-through node with input and output

Of course, one can design an infinite number of nodes, thus listing them all would be impossible.

If one is to increase the central node to more than 3 arms, one has one of two options. By connecting two central nodes using a pass-through node with zero travel time. One has essentially created a 4 armed central node. However, the division of goods per arm becomes less trivial. The other option is to model one from scratch. For this, it is required to know which vehicle arrives first, second, etc. Determining this using only maximisation and minimisation is possible, but it can quickly increase in size when the number of vehicles increases.

One option is to use the following expression to extract the k^{th} smallest value from a finite set X using only min and max, where X is the set of all relevant arrival times:

$$\mathrm{s}_k(X) = \max_{\substack{S \subseteq X \\ |S| = n-k+1}} \min(S) \tag{7-60}$$

This formula returns the k^{th} smallest element in the set $X = \{x_1, \ldots, x_n\}$ without explicit sorting. The idea is that every subset of size $n - k + 1$ must contain at least one of the k smallest elements. The smallest element in such a subset can therefore be no larger than $x_{(k)}$, and the subset that just includes $x_{(k)}$ (but none of the larger values) will yield it as the minimum. Taking the maximum over all minima returns $x_{(k)}$.

Example 7.2. *(Obtain k^{ht} smallest value using only* max *and* min *operations)*
Let $X = \{7, 4, 5, 1\}$ and suppose we want the third smallest value ($k = 3$). Then we take all subsets of size $n - k + 1 = 2$:

$$\begin{aligned} &\min\{7,4\} = 4 \quad \min\{7,5\} = 5 \quad \min\{7,1\} = 1 \\ &\min\{4,5\} = 4 \quad \min\{4,1\} = 1 \quad \min\{5,1\} = 1 \end{aligned} \tag{7-61}$$

Taking the maximum over these values gives $\mathrm{s}_3(X) = \max\{4, 5, 1, 4, 1, 1\} = 5$*, which is indeed the third smallest element.*

Currently, all nodes are designed to support only a single vehicle operating on a given route at a time. However in practice, one might want to allow for several vehicles to run on the same route. This will require complete remodelling of the system and nodes depending on the intended operational behaviour. For example, introducing minimum headways between vehicles and extra states to differentiate between different vehicles. This might be useful when goods are transferred to smaller vehicles, as the capacity must remain similar so as not to run into stability issues.

Extending the node framework to accommodate additional arms or multiple vehicles introduces significant complexity but also provides a pathway toward more realistic and flexible transport network models. While simple approaches like combining nodes can increase the number of arms, more accurate modelling requires detailed ordering of vehicle arrivals and advanced handling of interactions such as minimum headways in the case of multi-vehicle routes.

7-4 Generating the System of Equations from a Transport Graph

The previous section has introduced several different transport nodes and has also briefly touched upon more complex nodes and how to further extend this framework. In this section, an algorithm is introduced that describes how a transport network composed of various node types can be translated into a set of equations that define the system dynamics. The network consists of nodes stored in the database, including input nodes, output nodes, transfer nodes, pass-through nodes, and the central node. Each node corresponds to a subsystem, and arcs between nodes represent the movement of vehicles and goods.

It begins by assigning vehicle cycles and assembling a high-level structure of the system. This is followed by a detailed breakdown of the algorithm used to transform the graph into system matrices. Appendix D provides the full MATLAB implementation of this process.

7-4-1 Cycle Assignment and High-level System Assembly

Now that several types of node structures and subsystems have been introduced, the next step is to construct a full transport system by connecting these components. This section explains how the overall system model is built, how vehicle cycles are determined, and what considerations must be made during system construction.

The algorithm allows users to define a system using a high-level input consisting of an adjacency matrix and associated node properties. The adjacency matrix encodes the network structure: it is a symmetric matrix with entries of 1 indicating the presence of an arc between two nodes, and 0 otherwise. Since the system is undirected, an arc from node i to node j implies an arc from j to i as well, and thus the matrix is symmetric by construction.

The node properties include information such as travel times, node types, and flow capacities. If node types are not provided, the user will be prompted to assign them manually. The algorithm also validates vehicle information, which is especially important when pass-through nodes are present. In such cases, the number of effective vehicles on a route may not be immediately obvious. The validation step ensures that the number and routing of vehicles are consistent with the overall system structure.

Once the input is validated, the algorithm generates a complete MMPS system description. This model can then be analysed or simulated directly.

A challenging part of system construction is determining where cycle updates occur. Each vehicle follows a cyclic route, and its state must be updated at the right moment to indicate the end of one cycle and the beginning of the next. When a vehicle returns to its starting node, a new cycle begins. For standard routes without pass-through nodes, cycle update points are relatively easy to define and add. Every node is assigned an index based on the adjacency matrix. The rule applied is that a cycle update occurs when a vehicle arrives at the lower-numbered node in the cycle. This ensures that each cycle is updated only once. However, this rule must be adjusted when pass-through nodes are present in a given route. In these routes, going to a lower-number node triggering a cycle update is no longer sufficient, as multiple cycle updates can occur in a single route. To fix this, the cycle update is defined to occur upon arrival at the lowest-numbered end node of the route. This approach ensures that routes with pass-through nodes only have a single cycle update.

7-4-2 System Construction from a Transport Graph

This section outlines the process of converting a transport network into a system of equations. Each node in the network corresponds to a subsystem, and arcs between nodes represent vehicles. Based on the graph structure, node types and the sub-system matrices. Appendix D Shows the full Matlab implementation.

The algorithm can be broken down into 4 main stages:

1. Graph and node inputs
2. Truck assignment
3. Parameter filling and block gathering

4. Matrix construction and cycle management

The algorithm describing this process is given below:

Algorithm 5 System construction from a transport graph

Input: Graph adjacency matrix, node data with types and parameters
Output: Complete system matrices (A, B, C, D)

procedure BUILDTRANSPORTSYSTEM
 Validate input:
 - Check if the Adjacency matrix is executable with the current database
 - Ensure node types are correct and required parameters are present

 Assign trucks to arcs:
 - For each undirected arc (i, j), assign a unique truck
 - If node j is a pass-through, treat arcs $(i \to j)$ and $(j \to k)$ as a single truck route

 Attach truck parameters to nodes: For each truck, assign capacity, loading, and unloading rates

 Fill local system templates:
 - For each node, retrieve symbolic matrices from the template database (e.g., A, B, C, D, E_k, F, H, K, L)
 - Replace symbolic entries with node and truck parameters

 Assemble global matrices:
 - Build block-diagonal matrices A_{big}, B_{big}, C_{big}, D_{big}, F_{big}, H_{big}

 Construct communication matrix E:
 - For each connection between nodes, place the corresponding E_k block in E such that it aligns the connected arms

 Determine cycle update rules via K and L:
 - For pass-through connections, update the cycle at the lowest-numbered endpoint
 - For regular nodes, place KL blocks in K or L based on the direction (e.g., toward the lower-numbered node)

 Concatenate final system:

$$A_{\text{full}} = \begin{bmatrix} A_{\text{big}} & 0 \\ 0 & F_{\text{big}} \end{bmatrix}, \quad B_{\text{full}} = \begin{bmatrix} B_{\text{big}} & 0 \\ 0 & H_{\text{big}} \end{bmatrix}, \quad C_{\text{full}} = \begin{bmatrix} C_{\text{big}} & 0 \\ K & 0 \end{bmatrix}, \quad D_{\text{full}} = \begin{bmatrix} D_{\text{big}} & E \\ L & 0 \end{bmatrix}$$

end procedure

To illustrate how the algorithm operates in practice, the following example shows the construction of a system based on a simple transport network. This example highlights how the proposed individual building blocks are concatenated into a complete symbolic system.

Example 7.3. *(3 node system example using Algorithm 5)*
Consider a transportation system of 3 nodes with 1 input node with an inflow of γ, 1 transfer

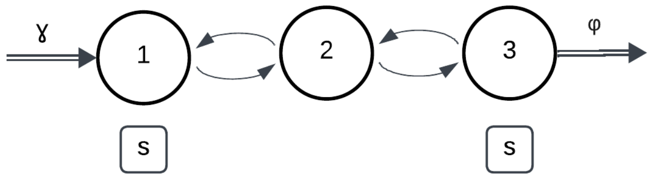


Figure 7-11: Graphical representation of the example system with 3 nodes

node without a stack and and 1 output node with an outflow of φ. This system can be visualised, as can be seen in Figure 7-11. It can also be represented as an adjacency graph as such:

$$G = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{7-62}$$

When using Algorithm 5, G is given as a first input. Then the node information is gathered, where the user in this example inputs the following parameters:

- *Node 1: type = input node, Inflow rate = 5, travel time to node = 10*
- *Node 2: type = transfer node without stack, travel time to node from connection 1 = 10, travel time to node from connection 2 = 8*
- *Node 3: type = output node, outflow rate = 5, travel time to node = 8*

After which, the number of vehicles will be calculated, which in this example is 2. The next step requires input of the vehicle parameters, which are, for example given as follows:

- *Vehicle 1: capacity = 50, loading rate = 40, unloading rate = 25*
- *Vehicle 2: capacity = 50, loading rate = 35, unloading rate = 30*

This is then used by Algorithm 5 to give system matrices. However, they are too large to show here, so they have been moved to Appendix B. The matrices are given by $A_{full} \in \mathbb{R}_{\varepsilon}^{24\times 27}$, $B_{full} \in \mathbb{R}_{\top}^{27\times 28}$, $C_{full}, D_{full} \in \mathbb{R}^{28\times 24}$. *These are verified to be the correct matrices.*

Chapter 8

Case Study: Transportation System

In this chapter, a transportation system of 4 nodes and 3 trucks is modelled, simulated and analysed. In this chapter, all concepts, techniques and new insights of the previous chapter, such as periodicity, the MILP algorithm and the modelling nodes are applied. Since the URS currently is the only working application of MMPS systems, this chapter introduces a transportation system modelled as an MMPS system. Section 8-1 introduces the basis of the system, with some constraints and assumptions. Section 8-2 derives the entire model, while explaining why specific choices are made. Section 8-3 finishes the chapter, where the transportation model is used for analysis and simulation. First, the system parameters are chosen, after which the growth rate and fixed points of the system are identified. The system is simulated, and the stability of the system is investigated, including the calculation of the maximal invariant set.

8-1 Introduction to a 4-Node Transportation System

This section introduces a 4-node transportation system served by three trucks. The nodes are connected as can be seen in Figure 8-1. Truck 1 drives between nodes 1 and 4, truck 2 drives between nodes 2 and 4, and truck 3 drives between nodes 3 and 4. There is an inflow of goods at node 1 and an outflow of goods at nodes 2 and 3. The goods do not have a fixed destination, so they can be brought to either node 2 or node 3. At node 4, goods are transferred from truck 1 to either truck 2 or truck 3, such that when truck 1 leaves, all the goods have been transferred to truck 2 and 3. When a truck arrives full, it will leave again once it is empty. When a truck arrives empty, it will either leave when it is full or when there are no more goods to take. At nodes 1, 2 and 3, there are storage stacks available. This system will be modelled as a DE MMPS system, where the state $x(k)$ denotes the time at which the k^{th} event has occurred. Notice that this is different from the URS in the case study in [5], where k denotes the train number. For this system to work, it is important that the capacity of truck 1 is not larger than the capacity of trucks 2 and 3 combined. Otherwise, truck 1 will never leave node 4, resulting in a deadlock.

Several system parameters are defined below:

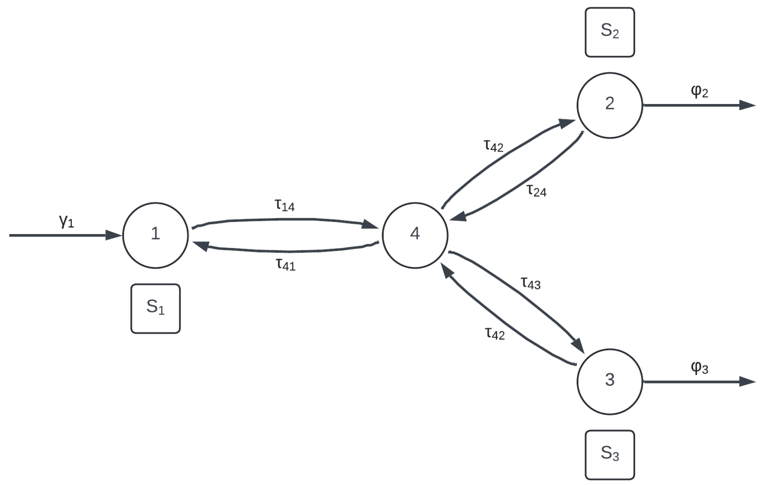


Figure 8-1: Graphical representation of the 4-node 3-truck system

- The capacity of truck i is given by C_i for $i \in \{1, 2, 3\}$
- The capacity of truck 1 is less than or equal the capacity of trucks 2 and 3 combined. i.e. $C_1 \leq C_2 + C_3$
- The inflow of goods per unit of time at node 1 is given by γ_1
- The outflow of goods per unit of time at nodes 2 and 3 is given by φ_2 and φ_3 respectively
- The travel time from node j to node k is denoted by τ_{jk}
- The unloading and loading speeds of truck i are given by u_i and L_i respectively

On top of that, some assumptions must be made before modelling this system, which are;

- It is assumed that $L_1 > \gamma_1$
- It is assumed that $u_2, u_3 > \gamma_1$
- There is no storage stack at node 4, so the transfer of goods can only start once truck 1 and truck 2 or 3 have arrived.

It is assumed that it is unknown what the capacities of the trucks are; the same holds for the loading and unloading speeds. This introduces an extra layer of complexity compared to the models from Chapter 7, as it is not known a priori whether the loading or unloading speed is the limiting factor during goods transfer. This will drastically increase the size of the system; however, it allows full flexibility in adjusting all parameters.

At node 4, there are three trucks that will arrive at some point. The arrival time of truck 1 coming from node 1 arriving at node 4 is given by $a_{41}(k)$. The arrival time of truck 2 coming from node 2 arriving at node 4 is given by $a_{42}(k)$. Lastly, the arrival time of truck 3 coming from node 3 arriving at node 4 is given by $a_{43}(k)$. The arrival time of truck 1 coming from node 4, going back to node 1, is given by $a_1(k)$. For the other two trucks, the same holds. The arrival time of truck 2 coming from node 4 going back to node 2 is given by $a_2(k)$, and the arrival time of truck 3 coming from node 4 going back to node 3 is given by $a_3(k)$.

For the departure times, the same notation is used. So the departure $d_1(k)$ represents the departure from node 1 towards node 4, and $d_{41}(k)$ represents the departure from node 4 towards node 1. Similarly, $d_2(k)$ and $d_{42}(k)$ correspond to departures between nodes 2 and 4, and $d_3(k)$ and $d_{43}(k)$ correspond to those between nodes 3 and 4.

The number of goods in the truck at departure is tracked. The number of goods in truck 1 at departure at node 1 is given by $\rho_1(k)$ and the number of goods in trucks 2 and 3 at departure at node 4 is given by $\rho_{42,\text{real}}(k)$ and $\rho_{43,\text{real}}(k)$ respectively. At any other departure, the trucks are empty, so these will not need to be tracked. During the modelling and simulation, it was discovered that in some very specific cases, the calculated truck loads were off, which is why in those cases the calculated truck load $\rho_{42,calc}(k)$ and $\rho_{43,calc}(k)$ needed to be repaired. Depending on system properties such as loading speeds and capacities, the system can be in one of three modes. This means that this system is technically a switching MMPS system, and using Section 7-2, the system will be modelled as a single MMPS system. This means that there are three extra states that indicate which mode the system is currently operating in. These states are $c_1(k)$, $c_2(k)$ and $c_3(k)$, where $c_i(k)$ indicates whether mode i is active or not. In total, the system will have 51 states: 12 time states, 6 quantity states, 3 mode states and 30 supporting states. These supporting states could have all been integrated into each proper state. however, for modelling purposes, it was chosen not to do so.

Before modelling the entire system, it is helpful to define some auxiliary states that will simplify later formulations. These auxiliary states are part of the supporting states and will occur so often that it is good to introduce them first. Specifically, the following are introduced:

- $L_f(k)$: the time at which the **first** truck can start loading in cycle k.
- $L_s(k)$: the time at which the **second** truck can start loading in cycle k.
- $a_f(k)$: the **arrival time** of the first truck that will receive goods in cycle k.
- $a_s(k)$: the **arrival time** of the second truck that will receive goods in cycle k.

These additional states serve as a foundation for modelling the interactions between arrivals and loading processes later in the system dynamics and are given as follows;

$$\begin{aligned} a_f(k) &= \min(a_{42}(k), a_{43}(k)) \\ a_s(k) &= \max(a_{42}(k), a_{43}(k)) \\ L_f(k) &= \max(a_{41}(k), \min(a_{42}(k), a_{43}(k))) \\ L_s(k) &= \max(a_{41}(k), a_{42}(k), a_{43}(k)) \end{aligned} \tag{8-1}$$

Notice that $L_f(k)$ and $a_f(k)$ and $L_s(k)$, and $a_s(k)$ will be the same if truck 1 arrives first; however will be different if truck 2 and or 3 arrives before truck 1.

8-2 Mathematical Derivation of the 4-Node Transportation System

In this section, the mathematical derivation of the 4-node transportation system is conducted. The section starts with deriving the equations for all the time states and derives a method for determining the different operating modes. After the time state, the quantity states are derived, after which the system is validated on time invariance and solvability.
The arrival time of a truck is given by the departure time at the previous node plus the travel time. One cycle of the system consists of all trucks travelling from a node and returning to their starting node. Since the trucks start at nodes 1,2, and 3, it means that their route also ends here, and a cycle update must take place. Thus, the arrival times at node 1, 2 and 3 are given by:

$$\begin{aligned} a_1(k) &= d_{41}(k-1) + \tau_{41} \\ a_2(k) &= d_{42}(k-1) + \tau_{42} \\ a_3(k) &= d_{43}(k-1) + \tau_{43} \end{aligned} \tag{8-2}$$

While the arrival times of the trucks at node 4 are given by:

$$\begin{aligned} a_{41}(k) &= d_1(k) + \tau_{14} \\ a_{42}(k) &= d_2(k) + \tau_{24} \\ a_{43}(k) &= d_3(k) + \tau_{34} \end{aligned} \tag{8-3}$$

Next, the departure times of all trucks can be modelled. This is very challenging and complex as the system has three different modes at node 4, which depend on the arrival times of the trucks at node 4, as well as the capacities and loading and unloading speeds.

- **Mode 1**: Both truck 2 and truck 3 arrive and start loading, but partway through, the available goods run out. Neither truck is fully loaded, but since there is nothing left to load, they both leave at the same time, so both are partially filled.
- **Mode 2**: One truck arrives after the other, but thanks to either a faster loading speed or smaller capacity, it finishes loading first and leaves before the truck that got there earlier.
- **Mode 3**: The first truck to arrive gets fully loaded and leaves. The second truck, arriving later, only gets a partial load because there are not enough goods left.

Figure 8-2 provides a visual breakdown of the conditions that lead to each of the three modes.

8-2-1 Determining the Active Mode

To model this behaviour correctly, Proposition 5 is used. To distinguish between the different modes of operation, mode state variables $c_1(k)$, $c_2(k)$, and $c_3(k)$ are introduced. Each $c_i(k)$ corresponds to mode i and is defined such that:

- $c_i(k) = 0$ if mode i is active at time step k,
- $c_i(k) < 0$ if mode i is not active at time step k.

This section focuses on defining the conditions under which each mode becomes active.

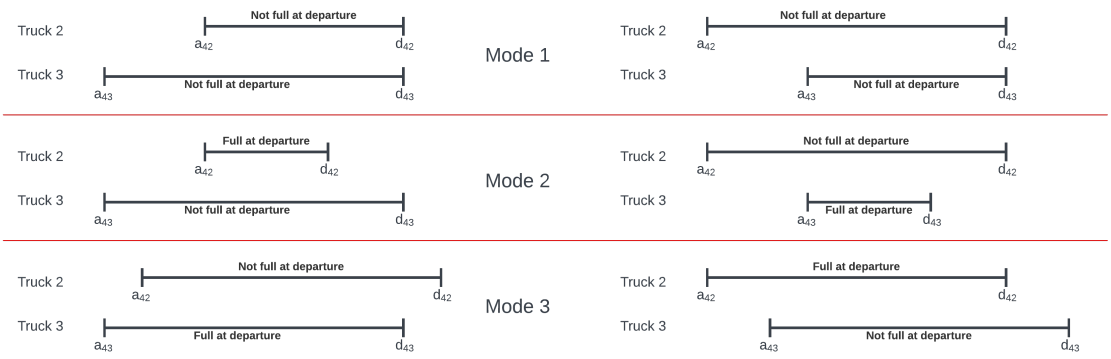


Figure 8-2: Graphical representation of the different modes of the transportation system

Characterising Activation of Mode 1

The goal is to derive an MMPS function for $c_1(k)$:

$$c_1(k) = f(x(k-1), x(k)) \tag{8-4}$$

where f is an MMPS function such that:

$$\begin{aligned} c_1(k) &= 0 \text{ if mode 1 is active} \\ c_1(k) &< 0 \text{ if mode 1 is not active} \end{aligned} \tag{8-5}$$

In order to achieve this, the function is split into several parts to make modelling and derivation easier. Mode 1 is active when all goods are placed in a truck without one of the two trucks becoming full. So it must be checked whether this is possible. To do so, first look at how many goods have been loaded between the time the first truck starts loading and when the second truck starts loading. Since it is not known in advance whether truck 1's unloading speed or the loading speed of the first arriving truck will be the bottleneck, the minimum of the two must be taken as goods cannot be loaded before they are unloaded. So the number of goods that have been loaded is given by:

$$(L_s(k) - L_f(k)) \cdot \min(l_f, u_1) \tag{8-6}$$

Where l_f refers to the load speed of the first arriving truck. Then the goods that are still left to be loaded are given by:

$$\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(l_f, u_1) \tag{8-7}$$

Then it must be checked that loading these goods in both trucks will not make any of them full. Since now two trucks are loading, the time it takes to load all present goods is less than when only 1 truck is loading. So the loading speed is given by:

$$\min(L_1 + L_2, L_1 + u_1, L_2 + u_1, 2u_1) \tag{8-8}$$

So the time all goods have been loaded in mode 1 is given by:

$$\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(l_f, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, 2u_1)} + L_s(k) \tag{8-9}$$

It must be checked that both truck 2 and truck 3 are not filled before this moment is reached, as then the system will not be in this mode. The time it takes to fill truck 2 and truck 3 completely are given by:

$$\begin{aligned} L_f(k) + \frac{C_f}{\min(l_f, u_1)} \\ L_s(k) + \frac{C_s}{\min(l_s, u_1)} \end{aligned} \tag{8-10}$$

Where C_f is the capacity of the first truck to be filled, C_s is the capacity of the second truck to start loading, and l_s is the load speed of the second truck to start loading. Notice that C_f and C_s are either C_2 or C_3 and not new capacities. Similarly, l_s and l_f are also already established loading speeds of either L_2 or L_3. It all depends on which truck arrives first and which arrives second.

Now to check whether no truck would be filled to capacity, the time it takes to load all goods in mode 1 given by (8-9) must be less than the time it takes to fill either one of the truck completely given by (8-10) This means that the following inequality constraints must hold when mode 1 is active:

$$\begin{aligned} \frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(l_f, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, 2u_1)} + L_s(k) \leq L_f(k) + \frac{C_f}{\min(l_f, u_1)} \\ \frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(l_f, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, , 2u_1)} + L_s(k) \leq L_s(k) + \frac{C_s}{\min(l_s, u_1)} \end{aligned} \tag{8-11}$$

It must also be checked that the first arriving truck has not taken so much that the second truck will not receive anything, which would mean the system is in mode 3. This is done by checking whether the departure time of the last truck to start loading, in the case where the last truck receives all remaining goods, is earlier than when both trucks load simultaneously. The time when the second arriving truck is done loading the remainder is given by:

$$\frac{\rho_1(k) - C_f}{\min(l_s, u_1)} + L_s(k) \tag{8-12}$$

Then the finishing time of the simultaneous loading is given by:

$$\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(l_f, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, 2u_1)} + L_s(k) \tag{8-13}$$

(8-13) must be larger than (8-12) since then the time when both trucks are done loading simultaneously is later than when the first arrival takes its capacity and the second takes the remainder, resulting in the following inequality condition:

$$\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(l_f, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, 2u_1)} + L_s(k) \geq \frac{\rho_1(k) - C_f}{\min(l_s, u_1)} + L_s(k) \tag{8-14}$$

These three conditions together, found in (8-11) and (8-14), are all true when mode 1 is active, but at least one is not when mode 1 is not active. To translate this into an MMPS function, the conditions are divided into two: one where truck 2 starts loading first and one where truck 3 starts loading first. This way, the variables C_f, C_s, l_f and l_s are known. Only the case where truck 2 arrives first will be discussed in detail, as the alternative case is analogous.

Thus now $C_f = C_2$, $C_s = C_3$, $l_f = L_2$ and $l_s = L_3$. Filling in C_f, C_s, l_f, l_s and rearranging (8-11) and (8-14) to the form:

$$\begin{aligned}
f_1(k) &= \frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(L_2, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, 2u_1)} + L_s(k) - L_s(k) - \frac{\rho_1(k) - C_2}{L_3} \\
f_2(k) &= -\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(L_2, u_1)}{\min(L_1 + L_2, l_f + u_1, 2u_1)} - L_s(k) + L_f(k) + \frac{C_2}{L_2} \\
f_3(k) &= -\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(L_2, u_1)}{\min(L_1 + L_2, l_f + u_1, 2u_1)} - L_s(k) + L_s(k) + \frac{C_3}{L_3}
\end{aligned} \tag{8-15}$$

This way, when $f_1(k)$, $f_2(k)$, and $f_3(k)$ are all larger than zero, mode 1 should be active, while when one is less than zero, mode 1 should not be active.

Then, by taking the minimum of $f_1(k)$, $f_2(k)$ and $f_3(k)$ together with $a_f(k) - a_{42}(k)$, we end up with a state that is zero when mode 1 is active and when truck 2 arrives first. Call this state $c_{1,\text{t2f}}$ where t2f stands for truck 2 first. $c_{1,\text{t2f}}$ is then given by:

$$c_{1,\text{t2f}} = \min(f_1(k) \cdot 10^5, f_2(k) \cdot 10^5, f_3(k) \cdot 10^5, a_f(k) - a_{42}(k)) \tag{8-16}$$

Notice that $a_f(k) - a_{42}(k)$ is always less than zero unless truck 2 arrives first, then it is exactly zero. Adding this makes sure that $c_{1,\text{t2f}}$ can only ever be equal to zero when truck 2 arrives first and is negative otherwise. Also, notice that $f_1(k)$, $f_2(k)$ and $f_3(k)$ have been multiplied with 10^5. This is to make the $c_{1,\text{t2f}}$ function steep such that when mode 1 is not active, at least one if $f_1(k)$, $f_2(k)$ and $f_3(k)$ is negative, thus by multiplying with 10^5 the entire $c_{1,\text{t2f}}$ function quickly become very negative, which is useful in obtaining the correct departure time.

The condition when mode 1 is active and truck 3 arrives first is denoted by $c_{1,\text{t3f}}$ is given by :

$$c_{1,\text{t3f}} = \min(f_4(k) \cdot 10^5, f_5(k) \cdot 10^5, f_6(k) \cdot 10^5, a_f(k) - a_{43}(k)) \tag{8-17}$$

With $f_4(k)$, $f_5(k)$, $f_6(k)$;

$$\begin{aligned}
f_4(k) &= \frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(L_3, u_1)}{\min(L_1 + L_2, L_3 + u_1, L_2 + u_1, 2u_1)} + L_s(k) - L_s(k) - \frac{\rho_1(k) - C_3}{L_2} \\
f_5(k) &= -\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(L_3, u_1)}{\min(L_1 + L_2, l_f + u_1, 2u_1)} - L_s(k) + L_f(k) + \frac{C_3}{L_3} \\
f_6(k) &= -\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot \min(L_3, u_1)}{\min(L_1 + L_2, l_f + u_1, 2u_1)} - L_s(k) + L_s(k) + \frac{C_2}{L_2}
\end{aligned} \tag{8-18}$$

If either $c_{1,\text{t2f}}$ or $c_{1,\text{t3f}}$ is equal to zero, mode 1 is active; the other one will then by definition be less than zero. By taking the maximum, it makes sure that if one is active, it will be detected. Resulting in condition for $c_1(k)$:

$$\begin{aligned}
c_1(k) = \max\Big(&\min\left(f_1(k) \cdot 10^5, f_2(k) \cdot 10^5, f_3(k) \cdot 10^5, (a_f(k) - a_{42}(k)) \cdot 10^8\right), \\
&\min\left(f_4(k) \cdot 10^5, f_5(k) \cdot 10^5, f_6(k) \cdot 10^5, (a_f(k) - a_{43}(k)) \cdot 10^8\right)\Big)
\end{aligned} \tag{8-19}$$

Note that the condition $(a_f(k) - a_{4i}(k))$ has been scaled by a factor of 10^8. This is again to ensure that $c_1(k)$ is negative when mode 1 is not active.

Characterising Activation of Mode 2

It is easier to derive the condition for mode 2. This mode is active when the truck that starts loading second is fully loaded before the first truck has finished loading the rest of the goods. The goal is to derive an MMPS function for $c_2(k)$:

$$c_2(k) = f(x(k-1), x(k)) \tag{8-20}$$

where f is an MMPS function such that:

$$\begin{aligned} c_2(k) &= 0 \text{ if mode 2 is active} \\ c_2(k) &< 0 \text{ if mode 2 is not active} \end{aligned} \tag{8-21}$$

The time when the second truck that starts loading, is full is given by the time at which loading starts plus the capacity of that truck divided by the loading speed, which is given by:

$$\frac{C_s}{\min(l_s, u_1)} + L_s(k) \tag{8-22}$$

The time it takes the other truck is given by the remaining goods $\rho_1(k) - C_s$ divided by the loading speed of that truck plus the time when loading started. This is given by:

$$L_f(k) + \frac{\rho_1(k) - C_s}{\min(l_f, u_1)} \tag{8-23}$$

Thus, when the following holds, mode 2 should be active:

$$\frac{C_s}{\min(l_s, u_1)} + L_s(k) \leq L_f(k) + \frac{\rho_1(k) - C_s}{\min(l_f, u_1)} \tag{8-24}$$

In the same way as for $c_1(k)$, this condition is divided into two separate parts, one where truck 2 starts loading first and one where truck 3 starts loading first. This way, the variables C_f, C_s, l_f and l_s are known. Only the case for truck 2 arriving first will be discussed in detail, since the other case is very similar. So now $C_f = C_2$, $C_s = C_3$, $l_f = L_2$ and $l_s = L_3$. Rewrite (8-24) into the form $f_7(k) \geq 0$ as such:

$$f_7(k) = L_f(k) + \frac{\rho_1(k) - C_3}{\min(L_2, u_1)} - \frac{C_3}{\min(L_3, u_1)} - L_s(k) \geq 0 \tag{8-25}$$

Thus, when $f_7(k)$ is larger than zero, mode 2 should be active, and when $f_7(k)$ is less than zero, mode 2 will not be active. In the same way as for $c_1(k)$, one can construct $c_{2,\text{t2f}}$ by taking the minimum of $f_7(k)$ and $a_f(k) - a_{42}(k)$ while multiplying with 10^5 and 10^8 respectively to steepen the slope of the condition as such:

$$c_{2,\text{t2f}} = \min(f_7(k) \cdot 10^5, (a_f(k) - a_{42}(k)) \cdot 10^8) \tag{8-26}$$

For the case where truck 3 is loading first, the same thing can be done as such:

$$f_8(k) = L_f(k) + \frac{\rho_1(k) - C_2}{\min(L_3, u_1)} - \frac{C_2}{\min(L_2, u_1)} - L_s(k) \geq 0 \tag{8-27}$$

$$c_{2,\mathrm{t3f}} = \min(f_8(k) \cdot 10^5, (a_f(k) - a_{43}(k)) \cdot 10^8) \tag{8-28}$$

If either $c_{2,\mathrm{t2f}}$ or $c_{2,\mathrm{t3f}}$ is equal to zero, mode 2 is active, and the other one will then by definition be less than zero, so taking the maximum makes sure that if one is active, it is detected. Resulting in condition for $c_2(k)$:

$$\begin{aligned} c_2(k) = \max\Big(&\min\left(f_7(k) \cdot 10^5, (a_f(k) - a_{42}(k)) \cdot 10^8\right), \\ &\min\left(f_8(k) \cdot 10^5, (a_f(k) - a_{43}(k)) \cdot 10^8\right)\Big) \end{aligned} \tag{8-29}$$

Characterising Activation of Mode 3

Mode 3 describes a situation where the first truck to begin loading is filled to its full capacity, while the second truck takes the remaining goods and leaves partially filled. The goal is to derive an MMPS function for $c_3(k)$:

$$c_3(k) = f(x(k-1), x(k)) \tag{8-30}$$

where f is an MMPS function such that:

$$\begin{aligned} &c_3(k) = 0 \text{ if mode 3 is active} \\ &c_3(k) < 0 \text{ if mode 3 is not active} \end{aligned} \tag{8-31}$$

Mode 3 is active when the first truck that starts loading is filled to its capacity, and the second truck that starts loading takes all the remaining goods. Thus, the activation condition for mode 3 looks very similar to that of mode 2, however, $L_f(k)$ and $L_s(k)$ have been switched, as the truck that starts loading first must take its full capacity, and the truck that starts loading second must take the remainder. This condition is given by:

$$L_f(k) + \frac{C_f(k)}{\min(u_1, l_f)} \leq L_s(k) + \frac{\rho_1(k) - C_f(k)}{\min(u_1, l_s)} \tag{8-32}$$

In the same way as for $c_1(k)$ and $c_2(k)$, this condition is divided into two separate scenarios: one where truck 2 starts loading first, and one where truck 3 starts loading first. This way, the variables C_f, C_s, l_f and l_s are known. Only the case for truck 2 arriving first will be discussed in detail since the other case is very similar. So now $C_f = C_2$, $C_s = C_3$, $l_f = L_2$ and $l_s = L_3$. Rewrite (8-24) into the form $f_9(k) \geq 0$ as such:

$$f_9(k) = L_s(k) + \frac{\rho_1(k) - C_3}{\min(L_2, u_1)} - \frac{C_3}{\min(L_3, u_1)} - L_f(k) \geq 0 \tag{8-33}$$

Therefore, when $f_9(k)$ is larger than zero, mode 3 should be active, and when $f_9(k)$ is less than zero, mode 3 should not be active. In the same way as for $c_1(k)$ and $c_2(k)$ can one construct $c_{3,\mathrm{t2f}}$ by taking the minimum of $f_9(k)$ and $a_f(k) - a_{42}(k)$, while multiplying with 10^5 and 10^8, respectively to steepen the slope of the condition, yielding the following condition:

$$c_{3,\mathrm{t2f}} = \min(f_9(k) \cdot 10^5, (a_f(k) - a_{43}(k)) \cdot 10^8) \tag{8-34}$$

For truck 3 loading first, the same thing can be done, yielding the following condition:

$$f_{10}(k) = L_s(k) + \frac{\rho_1(k) - C_2}{\min(L_3, u_1)} - \frac{C_2}{\min(L_2, u_1)} - L_f(k) \geq 0 \tag{8-35}$$

With

$$c_{3,\text{t3f}} = \min(f_{10}(k) \cdot 10^5, (a_f(k) - a_{42}(k)) \cdot 10^8) \tag{8-36}$$

If either $c_{3,\text{t2f}}$ or $c_{3,\text{t3f}}$ is equal to zero, mode 3 is active, and the other one will then by definition be less than zero. Thus, taking the maximum makes sure that if one is active, it is detected. Resulting in condition for $c_3(k)$:

$$\begin{aligned} c_3(k) = \max\Big(&\min\left(f_9(k) \cdot 10^5, (a_f(k) - a_{42}(k)) \cdot 10^8\right), \\ &\min\left(f_{10}(k) \cdot 10^5, (a_f(k) - a_{43}(k)) \cdot 10^8\right)\Big) \end{aligned} \tag{8-37}$$

In the special case that both trucks 2 and 3 arrive at the same time at node 4 and one of the trucks will be filled completely, something that can happen when the capacities of truck 2 and 3 are far apart, the system activates both mode 2 and mode 3. This is not correct and will be corrected with the use of a delta function. This delta function is used to assign the truck with the largest capacity to be virtually the first to arrive, which is then used to correct this error. Since a true Dirac delta function cannot be constructed as an MMPS function, an approximation is taken, which looks like a steep triangle function, which is given by:

$$\delta(k) = \max\left(0, \min(1 - (a_f(k) - a_s(k)) \cdot 10^8, 1 + (a_f(k) - a_s(k)) \cdot 10^8)\right) \tag{8-38}$$

This means that the delta function is equal to one when both trucks 2 and 3 arrive at node 4 at the same time. It is zero everywhere else, except when the arrival times are nearly identical; within a range of 10^{-8}. Then the delta function returns something between zero and one. This delta function, together with the difference in capacities, is integrated into the condition for mode 3 as follows:

$$\begin{aligned} c_3(k) = \max\Big(&\min\left(f_9(k) \cdot 10^5, f_9(k) \cdot 10^5 + (C_2 - C_3) \cdot \delta(k) \cdot 10^8, (a_f(k) - a_{42}(k)) \cdot 10^8\right), \\ &\min\left(f_{10}(k) \cdot 10^5, f_{10}(k) \cdot 10^5 + (C_3 - C_2) \cdot \delta(k) \cdot 10^8, (a_f(k) - a_{43}(k)) \cdot 10^8\right)\Big) \end{aligned} \tag{8-39}$$

In the case of simultaneous arrival, the goal is to have mode 2 active with the truck with the largest capacity marked as the first to start loading. This way, the truck with the smaller capacity will leave first and be full, while the other truck leaves last with the remaining goods, which also makes logical sense. However, in this case, mode 3 will have a false positive where the f_9 or f_{10} will detect the system to be in mode 3. This is the f function corresponding to the truck with the largest capacity arriving second. The term $(C_i - C_j) \cdot \delta(k) \cdot 10^8$ makes sure that when this happens, the false positive is suppressed, and mode 3 is deactivated. The other way around, $(C_j - C_i) \cdot \delta(k) \cdot 10^8$ is always positive, thus it never influences the other sub-conditions.

8-2-2 Deriving Truck Departure Times

With the three operating modes clearly defined, focus can be shifted to determining the corresponding departure times of truck 2 and truck 3 at node 4. These departure times vary depending on which mode is active, as each mode leads to different loading dynamics and completion conditions. For each mode, a specific expression is derived that captures how long the trucks spend at the node before continuing their journey.

Departure Time in Mode 1

In mode 1, both truck 2 and truck 3 are partially loaded and leave the node at the same time, since loading stops when the available goods run out. Denote their shared departure time at node 4 by $d_{4.1}(k)$. This departure time is given by the time at which the last truck starts loading, plus the time it takes to load all goods that are still left. The departure time depends on which truck starts loading first and which starts loading second. This is similar to the mode detection, where the arrival and loading times also influence how the active mode is detected. First, the general departure time will be given, then all possibilities based on the limiting loading/unloading speed will be given. The goods that have already been loaded onto the first truck before the second truck starts loading is given by:

$$(L_s(k) - L_f(k)) \cdot l_f \tag{8-40}$$

This means that the goods that are still left are given by:

$$\rho_1(k) - (L_s(k) - L_f(k)) \cdot l_f \tag{8-41}$$

Dividing this by the total loading speed of both trucks gives the total loading time. Then, adding this to the start time of when the second truck starts loading gives us the departure time:

$$\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot l_f}{\text{total load speed}} + L_s(k) \tag{8-42}$$

There are different possible scenarios of combinations of loading speeds by which the departure time can be calculated. The departure time is taken as the maximum of all 7 possible combinations, which are given in the table below:

l_f	total load speed	
L_2	$L_2 + L_3$	
u_1	$u_1 + L_3$	Truck 2 arriving first
L_2	$L_2 + u_1$	
u_1	$2 \cdot u_1$	Truck 2 or 3 arriving first
L_3	$L_2 + L_3$	
L_3	$u_1 + L_3$	Truck 3 arriving first
u_1	$L_2 + u_1$	

Table 8-1: All 7 different loading and unloading combinations in mode 1

To ensure only options with the correct truck order are considered, $(a_f - a_{4i}(k)) \cdot 10^8$ is used again. This steep term makes sure that all scenarios where the ordering of the trucks is not in line with the true ordering are pushed far down, making sure the maximum is never taken with a departure time with wrong ordering. This gives the final departure time for mode 1:

$$\begin{aligned} d_{4.1}(k) = \max\Big(&\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot L_2}{L_2 + L_3} + L_s(k) + (a_f(k) - a_{42}(k)) \cdot 1e8, \\ &\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot u_1}{u_1 + L_3} + L_s(k) + (a_f(k) - a_{42}(k)) \cdot 1e8, \\ &\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot L_2}{L_2 + u_1} + L_s(k) + (a_f(k) - a_{42}(k)) \cdot 1e8, \\ &\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot u_1}{2u_1} + L_s(k), \\ &\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot L_3}{L_2 + L_3} + L_s(k) + (a_f(k) - a_{43}(k)) \cdot 1e8, \\ &\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot L_3}{u_1 + L_3} + L_s(k) + (a_f(k) - a_{43}(k)) \cdot 1e8, \\ &\frac{\rho_1(k) - (L_s(k) - L_f(k)) \cdot u_1}{L_2 + u_1} + L_s(k) + (a_f(k) - a_{43}(k)) \cdot 1e8\Big) \end{aligned} \tag{8-43}$$

Departure Time in Mode 2

In mode 2, the departure of trucks 2 and 3 depends on which truck is the first that start loading or the second that start loading. So the departure time for truck 2 in mode 2 at node 4 ($d_{422}(k)$) is different from the departure time for truck 3 in mode 2 at node 4 ($d_{432}(k)$). However, the derivation is the same; the variables are changed to align with truck 3 instead of truck 2. That is why only the derivation for truck 2 will be done, after which both departure times are presented.

If truck 2 is the first to start loading, it takes all the goods that truck 3 does not take. These goods are $\rho_1(k) - C_3$. Then, dividing this by the loading speed and adding the time when loading starts gives us the time when truck 2 should leave:

$$L_f(k) + \frac{\rho_1(k) - C_3}{\min(L_2, u_1)} \tag{8-44}$$

When truck 2 is the second to start loading, it will be full at departure. It starts loading at $L_s(k)$ and will leave once the time has passed to fill to capacity. Therefore, this departure time is given by:

$$L_s(k) + \frac{C_2}{\min(L_2, u_1)} \tag{8-45}$$

To determine whether truck 2 starts loading first or second, the terms $(a_{42}(k) - a_s(k)) \cdot 10^8$ and $(a_f(k) - a_{42}(k)) \cdot 10^8$ are added to the corresponding departure times, and the maximum over all options is taken to obtain the correct departure time. This again pushed the departure time corresponding to the wrong arrival order down, making sure only the departure options of the correct arrival order are considered, leading to:

$$\begin{aligned}
d_{422}(k) = \max \Big(& L_s(k) + \frac{C_2}{u_1} + (a_{42}(k) - a_s(k)) \cdot 10^8 \\
& L_s(k) + \frac{C_2}{L_2} + (a_{42}(k) - a_s(k)) \cdot 10^8, \\
& L_f(k) + \frac{\rho_1(k) - C_3}{u_1} + (a_f(k) - a_{42}(k)) \cdot 10^8 \\
& L_f(k) + \frac{\rho_1(k) - C_3}{L_2} + (a_f(k) - a_{42}(k)) \cdot 10^8 \Big)
\end{aligned} \tag{8-46}$$

In the case of simultaneous arrival, this poses a problem. In this case both $(a_{42}(k) - a_s(k)) \cdot 10^8$ and $(a_f(k) - a_{42}(k)) \cdot 10^8$ are equal to zero and also for truck 3, it means that the max operator takes the departure time of a full truck for both of them. This is, of course, not what it desired. So the delta function of (8-38) can be used in the same way again by making sure the departure time of the truck with the largest capacity becomes infeasible. This is integrated into the departure time as follows:

$$\begin{aligned}
d_{422}(k) = \max \Big(& \min(L_s(k) + \frac{C_2}{u_1} + (a_{42}(k) - a_s(k)) \cdot 1e10, \\
& L_s(k) + \frac{C_2}{u_1} + (a_{42}(k) - a_s(k)) \cdot 10^8 + (C_3 - C_2) \cdot 10^8 \cdot \delta(k)), \\
& \min(L_s(k) + \frac{C_2}{L_2} + (a_{42}(k) - a_s(k)) \cdot 10^8, \\
& L_s(k) + \frac{C_2}{L_2} + (a_{42}(k) - a_s(k)) \cdot 10^8 + (C_3 - C_2) \cdot 1e8 \cdot \delta(k)), \\
& L_f(k) + \frac{\rho_1(k) - C_3}{u_1} + (a_f(k) - a_{42}(k)) \cdot 10^8 \\
& L_f(k) + \frac{\rho_1(k) - C_3}{L_2} + (a_f(k) - a_{42}(k)) \cdot 10^8 \Big)
\end{aligned} \tag{8-47}$$

The departure time of truck 3 in mode 2 is given by:

$$\begin{aligned}
d_{432}(k) = \max \Big(& \min \big(L_s(k) + \frac{C_3}{u_1} + (a_{43}(k) - a_s(k)) \cdot 10^8, \\
& L_s(k) + \frac{C_3}{u_1} + (a_{43}(k) - a_s(k)) \cdot 10^8 + (C_2 - C_3) \cdot 10^8 \cdot \delta(k)\big), \\
& \min \big(L_s(k) + \frac{C_3}{L_3} + (a_{43}(k) - a_s(k)) \cdot 10^8, \\
& L_s(k) + \frac{C_3}{L_3} + (a_{43}(k) - a_s(k)) \cdot 10^8 + (C_2 - C_3) \cdot 10^8 \cdot \delta(k)\big), \\
& L_f(k) + \frac{\rho_1(k) - C_2}{u_1} + (a_f(k) - a_{43}(k)) \cdot 10^8 \\
& L_f(k) + \frac{\rho_1(k) - C_2}{L_3} + (a_f(k) - a_{43}(k)) \cdot 10^8 \Big)
\end{aligned} \tag{8-48}$$

Departure Time in Mode 3

Just like in mode 2, in mode 3, the departure times of the first and second arriving trucks are different. Thus, the departure time for truck 2 in mode 3 at node 4, $d_{423}(k)$, is different from the departure time for truck 3 in mode 3 at node 4, $d_{433}(k)$. However, the derivation is exactly the same; the variables are changed to align with truck 3 instead of truck 2. That is why only the derivation for truck 2 will be done, after which both will be fully presented.

If truck 2 starts loading first, it will either take all the goods that truck 1 has brought if this is less than its capacity, or it will be filled to capacity. So this departure time is given by:

$$L_f(k) + \frac{\min(\rho_1(k), C_2)}{\min(u_1, L_2)} \tag{8-49}$$

When truck 2 starts loading last, it will take all the remaining goods, if any. If there are no remaining goods, truck 2 will immediately leave again. In that scenario, the departure time is given by the start loading time $L_s(k)$. Otherwise, it will take the remaining goods $\rho_1(k) - C_3$, divide it by the loading speed and add the start loading time to get the departure time as such:

$$L_s(k) + \frac{\rho_1(k) - C_3}{\min(L_2, u_1)} \tag{8-50}$$

To determine whether truck 2 starts loading first or second $(a_{42}(k) - a_s(k)) \cdot 10^8$ and $(a_f(k) - a_{42}(k)) \cdot 10^8$ are used by adding them to the corresponding departure time and taking the max over all options will give us the correct departure time. This again pushed the departure time corresponding to the wrong arrival order down, making sure only the departure options of the correct arrival order are considered, leading to:

$$\begin{aligned}
d_{423}(k) = \max\Big(& \min\big(L_f(k) + \frac{C_2}{L_2} + (a_f(k) - a_{42}(k)) \cdot 1e10, \\
& L_f(k) + \frac{\rho_1(k)}{L_2} + (a_f(k) - a_{42}(k)) \cdot 1e10\big), \\
& \min\big(L_f(k) + \frac{C_2}{u_1} + (a_f(k) - a_{42}(k)) \cdot 1e10, \\
& L_f(k) + \frac{\rho_1(k)}{u_1} + (a_f(k) - a_{42}(k)) \cdot 1e10\big), \\
& L_s(k) + (a_{42}(k) - a_s(k)) \cdot 1e10, \\
& L_s(k) + \frac{\rho_1(k) - C_3}{L_2} + (a_{42}(k) - a_s(k)) \cdot 1e10, \\
& L_s(k) + \frac{\rho_1(k) - C_3}{u_1} + (a_{42}(k) - a_s(k)) \cdot 1e10\Big)
\end{aligned} \tag{8-51}$$

The departure time of truck 3 in mode 3 is given by:

$$
\begin{aligned}
d_{433}(k) = \max\Big(&\min\big(L_f(k) + \frac{C_3}{L_3} + (a_f(k) - a_{43}(k)) \cdot 1e10, \\
&L_f(k) + \frac{\rho_1(k)}{L_3} + (a_f(k) - a_{43}(k)) \cdot 1e10\big), \\
&\min\big(L_f(k) + \frac{C_3}{u_1} + (a_f(k) - a_{43}(k)) \cdot 1e10, \\
&L_f(k) + \frac{\rho_1(k)}{u_1} + (a_f(k) - a_{43}(k)) \cdot 1e10\big), \\
&L_s(k) + (a_{43}(k) - a_s(k)) \cdot 1e10, \\
&L_s(k) + \frac{\rho_1(k) - C_2}{L_3} + (a_{43}(k) - a_s(k)) \cdot 1e10, \\
&L_s(k) + \frac{\rho_1(k) - C_2}{u_1} + (a_{43}(k) - a_s(k)) \cdot 1e10\Big)
\end{aligned}
\tag{8-52}
$$

Computation of Remaining Departure Times

Now that the mode-dependent behaviours are clearly defined, the final expressions for the departure times of all trucks can be derived. Some of these departure times depend on which mode is active and must be formulated in a way that selects the correct value for each case.

Using the state variables corresponding to each mode $c_1(k)$, $c_2(k)$, and $c_3(k)$, which are zero when their corresponding mode is active, and negative otherwise, using the maximisation operation, one can naturally select the correct departure time. This approach allows all possible mode-specific departure times to be encoded in a single equation for each truck.

The resulting expressions cover:

- Truck 2 and truck 3 departing from node 4 under different modes,
- Truck 1 departing from node 4 when all goods are removed,
- Truck 1 departing from node 1 based on either reaching full capacity or exhausting the available goods,
- Trucks 2 and 3 departing from nodes 2 and 3, respectively, once unloaded.

From $c_1(k)$, $c_2(k)$, $c_3(k)$, it is known that they are negative when that mode is not active and zero when that mode is active. By adding them to the different departure time options and taking the maximum, one ends up with the correct departure time given below:

$$\begin{aligned} d_{42}(k) = \max\Big(& d_{4.1}(k) + c_1(k), \\ & d_{422}(k) + c_2(k), \\ & d_{423}(k) + c_3(k)\Big) \\ d_{43}(k) = \max\Big(& d_{4.1}(k) + c_1(k), \\ & d_{432}(k) + c_2(k), \\ & d_{433}(k) + c_3(k)\Big) \end{aligned} \tag{8-53}$$

Truck 1 leaves node 1 when all goods have been removed, and so it leaves together with the departure of the last truck, which is either truck 2 or truck 3. The departure time for truck 1 at node 4 is given by:

$$d_{41}(k) = \max(d_{42}(k), d_{43}(k)) \tag{8-54}$$

This choice can introduce a small modelling error in cases where truck 1 does not deliver enough goods to fill even the first arriving outbound truck. In those situations, truck 1 still waits for the second truck to arrive before leaving, even if there is nothing to transfer between them.

In reality, this would not happen. If truck 1 cannot bring enough goods to load both outbound trucks meaningfully, there is no reason to send the second one at all. That kind of inefficient behaviour would be avoided in any practical setup.

By choosing appropriate system parameters, it is possible to prevent this edge case from happening in the model too, keeping the simulated behaviour in line with how things would actually work.

For the departure of truck 1 at node 1, there are 2 options: the truck will depart when there are no more goods in the stack, or when the truck is full.

When a truck is filled, the time it takes will be equal to $\frac{C_1}{L_1}$. This is added to the arrival time, resulting in the departure time of the full truck. To determine the departure time of truck 1 when all goods have been taken the goods we must first look at how many goods there are to take. All goods that can enter the truck are given by: $s_1(k) = s_1(k-1) + \gamma_1 (d_1(k) - d_1(k-1))$. What can enter is given by $L_1 (d_1(k) - a_1(k))$. Setting these two equations equal to one another will give the departure $d_1(k)$ when all goods can enter:

$$d_1(k) = (L_1 - \gamma_1)^{-1}(s_1(k-1) + L_1 a_1(k) - \gamma_1 d_1(k-1)) \tag{8-55}$$

Combining the two yields:

$$d_1(k) = \min\Big(a_1(k) + \frac{C_1}{L_1}, (L_1 - \gamma_1)^{-1} \cdot \big(s_1(k-1) + L_1 a_1(k) - \gamma_1 d_1(k-1)\big)\Big) \tag{8-56}$$

At nodes 2 and 3, the trucks will depart the moment they are empty. The time it takes to empty the truck is given by dividing the load by the unloading speed. Then, adding the arrival time will result in the departure time. The departure times of the trucks at nodes 2 and 3 are given by:

$$\begin{aligned} d_2(k) &= a_2(k) + \frac{\rho_{42,\text{real}}(k-1)}{u_2} \\ d_3(k) &= a_3(k) + \frac{\rho_{43,\text{real}}(k-1)}{u_3} \end{aligned} \tag{8-57}$$

8-2-3 Derivation of Quantity States for the 4-Node Transportation System

In addition to all the time states and mode behaviour, it is essential to track how goods move through the system over cycles. This includes the quantities loaded onto and unloaded from each truck, as well as the accumulation of goods in stacks at nodes 1, 2, and 3. In this section, the state variables that represent the amount of goods processed by the system, which are the truckloads and the stack levels at nodes 1, 2, and 3, are defined.

Determining the Truckloads for Each Truck

The quantity of goods loaded into each truck depends on its loading duration and speed. For truck 1 at node 1, the calculation is straightforward: the truckload is the product of the loading time and the loading speed. For truck 1 at node 1, this is given by:

$$\rho_1(k) = L_1 \cdot (d_1(k) - a_1(k)) \tag{8-58}$$

For trucks 2 and 3 at node 4, this is a bit more challenging. The time when loading starts is given by $\max(a_{41}(k), a_{42}(k))$ and $\max(a_{41}(k), a_{43}(k))$ respectively. Then, depending on whether the unloading speed of truck 1, or the loading speed of truck 2 or 3 is the bottleneck in goods transfer, the calculated truck loads for trucks 2 and 3 are given by:

$$\begin{aligned} \rho_{42,calc}(k) = \min\Big(&L_2 \cdot (d_{42}(k) - a_{42}(k)), \\ &L_2 \cdot (d_{42}(k) - a_{41}(k)), \\ &u_1 \cdot (d_{42}(k) - a_{42}(k)), \\ &u_1 \cdot (d_{42}(k) - a_{41}(k))\Big) \\ \rho_{43,calc}(k) = \min\Big(&L_3 \cdot (d_{43}(k) - a_{43}(k)), \\ &L_3 \cdot (d_{43}(k) - a_{41}(k)), \\ &u_1 \cdot (d_{43}(k) - a_{43}(k)), \\ &u_1 \cdot (d_{43}(k) - a_{41}(k))\Big) \end{aligned} \tag{8-59}$$

However, when the arrival times are very close together, in the order of 10^{-6} units of time difference, the mode detection has a slight round off error. This results in the system thinking that both trucks are full, resulting in the creation of goods. In order to correct this, $\rho_{42,calc}(k)$

and $\rho_{43,calc}(k)$ are checked and potentially compensated accordingly. As was previously mentioned, when this happens, the truck with the largest capacity must be corrected. Again, the delta function (8-38) is used. Since the delta function is a steep triangle function, it means that the delta function is non-zero when the arrival times are close together. This means that in a similar manner to $c_3(k)$, the delta function can be used. A new auxiliary state $\rho_{4i,\text{comp}}$ for $i = \{1, 2\}$ is used which will return the correct truck load for truck i if truck i has the largest capacity and will return a larger truck load if the capacity of truck i is the smaller of the two as such:

$$\begin{aligned} \rho_{42,\text{comp}}(k) &= \max(\rho_1(k) - \rho_{43,\text{calc}}(k), \rho_1(k) - \rho_{43,\text{calc}}(k) + (C_3 - C_2) \cdot \delta(k)) \\ \rho_{43,\text{comp}}(k) &= \max(\rho_1(k) - \rho_{42,\text{calc}}(k), \rho_1(k) - \rho_{42,\text{calc}}(k) + (C_2 - C_3) \cdot \delta(k)) \end{aligned} \tag{8-60}$$

Then by taking the minimum with the calculated truck loads $\rho_{42,\text{calc}}$ and $\rho_{43,\text{calc}}$, the wrong truckload is corrected while the correct truck load remains unaffected as follows:

$$\begin{aligned} \rho_{42,\text{real}}(k) &= \min(\rho_{42,\text{calc}}(k), \rho_{42,\text{comp}}(k)) \\ \rho_{43,\text{real}}(k) &= \min(\rho_{43,\text{calc}}(k), \rho_{43,\text{comp}}(k)) \end{aligned} \tag{8-61}$$

This wrong truckload can be attributed to a wrongly chosen departure time, which means that it has an effect that the truck with the largest capacity will have to wait unnecessarily at node 4. However, this will result in the arrival times being further apart in the next cycle.

Determining the Stack Sizes at Departure

The stack at each node stores goods temporarily between truck arrivals and departures. These stacks act as buffers that absorb differences in timing or speed between input and loading processes and output and unloading processes. The number of goods in the stack at node 1 $s_1(k)$ is given by what was there previous cycle plus what was taken out and or brought in. What was put in via the input is given by

$$\gamma_1(d_1(k) - d_1(k-1)) \tag{8-62}$$

What was taken out is given by:

$$L_1(d_1(k) - a_1(k)) \tag{8-63}$$

Combining this plus what was left in the stack from the previous cycle results in an equation for the stack size at node 1:

$$s_1(k) = s_1(k-1) + \gamma_1\left(d_1(k) - d_1(k-1)\right) - L_1\left(d_1(k) - a_1(k)\right) \tag{8-64}$$

The stacks at nodes 2 and 3, $s_2(k), s_3(k)$, can be determined similarly. However, the way goods are added and removed from the stack is reversed. Thus, the goods that are added to the stack are given by:

$$\begin{aligned} &u_2(d_2(k) - a_2(k)) \\ &u_3(d_3(k) - a_3(k)) \end{aligned} \tag{8-65}$$

The number of goods that are taken out of the stack are given by:

$$\begin{aligned} &\varphi_2(d_2(k) - d_2(k-1)) \\ &\varphi_3\left(d_3(k) - d_3\left(k-1\right)\right) \end{aligned} \tag{8-66}$$

For these stacks, it is also important to make sure they never become negative, since this is not physically possible, and so the maximum with zero must be taken to ensure this, resulting in the following equations for $s_2(k)$ and $s_3(k)$:

$$\begin{aligned} s_2(k) &= \max\Big(0, s_2(k-1) - \varphi_2\left(d_2(k) - d_2\left(k-1\right)\right) + u_2(d_2(k) - a_2(k))\Big) \\ s_3(k) &= \max\Big(0, s_3(k-1) - \varphi_3\left(d_3(k) - d_3\left(k-1\right)\right) + u_3(d_3(k) - a_3(k))\Big) \end{aligned} \tag{8-67}$$

8-2-4 Model Validation

The state space equations for the transportation system given in Section 8-2 are hard to work with. Since it is an implicit MMPS system, obtaining the dependencies for each state can be tedious. The transportation system can be written into the MMPS ABCD canonical form. This allows for easy validation, analysis and simulation. Matrices A, B, C, and D are too large to depict here. However, they have been constructed in the following shape:

$$\begin{aligned} \begin{bmatrix} x_{\mathrm{t}}(k) \\ x_{\mathrm{q}}(k) \end{bmatrix} = \underbrace{\begin{bmatrix} A_{\mathrm{t}} & \varepsilon \\ \varepsilon & A_{\mathrm{q}} \end{bmatrix}}_{A} \otimes \Bigg(\underbrace{\begin{bmatrix} B_{\mathrm{t}} & \top \\ \top & B_{\mathrm{q}} \end{bmatrix}}_{B} \otimes' \Bigg(\underbrace{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{C} \cdot \begin{bmatrix} x_{\mathrm{t}}(k-1) \\ x_{\mathrm{q}}(k-1) \end{bmatrix} \\ + \underbrace{\begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix}}_{D} \cdot \begin{bmatrix} x_t(k) \\ x_q(k) \end{bmatrix} \Bigg)\Bigg) \end{aligned} \tag{8-68}$$

Where $A_t \in \mathbb{R}_{\varepsilon}^{37\times 76}$, $A_q \in \mathbb{R}_{\varepsilon}^{14\times 21}$, $B_t \in \mathbb{R}_{\top}^{76\times 97}$, $B_q \in \mathbb{R}_{\top}^{21\times 24}$, $C_{11}, D_{11} \in \mathbb{R}^{97\times 37}$, $C_{12}, D_{12} \in \mathbb{R}^{97\times 14}$, $C_{21}, D_{21} \in \mathbb{R}^{24\times 37}$ and $C_{22}, D_{22} \in \mathbb{R}^{24\times 14}$. The state vector $x(k)$ is given by:

$$\begin{aligned} x(k) = \big[\, & a_1(k),\ a_2(k),\ a_3(k),\ a_{41}(k),\ a_{42}(k),\ a_{43}(k),\ a_f(k),\ a_s(k),\ L_f(k),\ L_s(k), \\ & d_1(k),\ d_2(k),\ d_3(k),\ d_{41}(k),\ d_{42}(k),\ d_{43}(k),\ d_{4.1}(k),\ d_{422}(k),\ d_{423}(k), \\ & d_{432}(k),\ d_{433}(k),\ f(k),\ c_1(k),\ c_2(k),\ c_3(k),\ s_1(k),\ s_2(k),\ s_3(k), \\ & \rho_1(k),\ \rho_{42,\mathrm{calc}}(k),\ \rho_{43,\mathrm{calc}}(k),\ \delta(k),\ \rho_{42,\mathrm{comp}}(k),\ \rho_{42,\mathrm{real}}(k), \\ & \rho_{43,\mathrm{comp}}(k),\ \rho_{43,\mathrm{real}}(k)\,\big]^{\top} \end{aligned} \tag{8-69}$$

Where $f(k) \in \mathbb{R}^{16}$ consists of all subparts of the mode detection states f_i for $i = \{1, \dots, 16\}$.

Since the system contains a finite D matrix, the system is implicit. To check whether the system is time invariant, and any of the analysis methods can be applied, the conditions of (3-7) must hold, which can be found below as well. When this is true, it can be concluded that the system is time-invariant.

$$\sum_{i \in \overline{n_{\mathrm{t}}}} [C_{11}D_{11}]_{\ell i} = 1, \forall \ell \in \overline{p_{\mathrm{t}}}, \quad \sum_{i \in \overline{n_{\mathrm{t}}}} [C_{21}D_{21}]_{ti} = 0, \forall t \in \overline{p_{\mathrm{q}}} \tag{8-70}$$

Because the sub-matrices of C and D are too large, they have been omitted. However, after verification of (3-7), it can be seen that this condition holds, and thus it can be concluded that the system is time invariant. Validating this result can be done by using the code provided in Appendix E, where sub-matrices C_{11}, D_{11}, C_{21} and D_{21} are identified and the validity of (8-70) is checked.

Secondly, the solvability of the system is checked. By transforming the A, B and D matrices into structure matrices S_A, S_B, S_D respectively, following [5]. For the system to be solvable, there must exist a transformation matrix T such that $F = T \cdot S_A \cdot S_B \cdot S_D \cdot T^{-1}$ is strictly lower triangular. This condition is equivalent to checking if there are any cycles present in the communications graph of $S_\otimes$, which is defined as:

$$[S_\otimes]_{i,j} = \begin{cases} [S]_{i,j} \text{ if } [S]_{i,j} \neq 0 \\ \varepsilon \text{ if } [S]_{i,j} = 0 \end{cases} \tag{8-71}$$

where $S = S_A \cdot S_B \cdot S_D$ [17]. This matrix is too large to show on a page. However, the communications graph of $S_\otimes$ does not contain any cycles. From this, it can be concluded that the system is solvable.

Finally, some last remarks about the system. The values 10^5 and 10^8 are used as they are large enough to achieve their goal with the used variable sizes, while also leaving enough room in the machine precision to accurately calculate the system states. Picking a gain that is too high can cause numerical issues. MATLAB, where the simulation runs, has a machine precision of 16 digits. Thus, if the gain is too large, most of those digits get used up by the big values, leaving nothing to capture the small differences in departure times.

8-3 Simulation and Analysis of the 4-Node Transportation Network

Now that the system is modelled and validated, it can be simulated and analysed which will be done in this Section. First, the system parameters will be chosen, after which the growth rate, fixed point and periodic orbits will be determined. Then, the system will be initialised in these fixed points and periodic orbits, after which the stability and invariant set for all different dominant modes will be determined.

8-3-1 Initialisation of the System

In order to simulate and analyse the transportation system, it must be initialised. Initialising the system is relatively straightforward. All state information must be present and correct, and the system parameters must be determined. The chosen system parameters can be found in (8-72).

$$\begin{aligned}
&\textbf{In- and output flow rates:}\\
&\quad \gamma_1 = \frac{75}{11.75}, \quad \varphi_2 = 5, \quad \varphi_3 = 5\\
&\textbf{Loading speeds:}\\
&\quad L_1 = 40, \quad L_2 = 20, \quad L_3 = 20\\
&\textbf{Unloading speeds:}\\
&\quad u_1 = 40, \quad u_2 = 20, \quad u_3 = 20\\
&\textbf{Travel times:}\\
&\quad \tau_{14} = 2, \quad \tau_{41} = 2, \quad \tau_{24} = 4, \quad \tau_{42} = 4, \quad \tau_{34} = 4, \quad \tau_{43} = 4\\
&\textbf{Capacities:}\\
&\quad C_1 = 75, \quad C_2 = 50, \quad C_3 = 50
\end{aligned} \tag{8-72}$$

The assumptions introduced in Section 8-1 are respected throughout this analysis. While the numerical values of the system parameters can be adjusted to some extent, these stability conditions must be satisfied to ensure the system remains operable and can evolve without congestion or deadlocks. These conditions guarantee that the system can operate in a consistent, repeatable manner without infinite accumulation of goods or blocked vehicle movements. Meaning the system operates in a stable manner.

The following conditions serve as practical stability requirements for the 4-node system:

- $C_1 < C_2 + C_3$: Ensures that node 4 does not retain any goods after each delivery cycle. This condition guarantees that vehicle 1 can always return to node 1 without encountering a deadlock due to residual goods not being able to fit in truck 1 or 2 in the same cycle.
- $\gamma_1 \ll L_1$: Implies that the arrival rate of incoming goods is significantly lower than the loading capacity at node 1. This allows for smooth intake and prevents build-up at the source.

- $\gamma_1 \ll u_2, u_3$: Ensures that the unload rates at nodes 2 and 3 are significantly higher than the arrival rate of goods into the system. This condition allows goods to be processed and forwarded through the system without a build-up of goods and a stack increasing towards infinity.
- $\gamma_1 \leq \varphi_2 + \varphi_3$: Guarantees that the total outflow capacity from nodes 2 and 3 is at least as large as the inflow rate at node 1. Without this condition, a continuous accumulation of goods in the stacks would occur, eventually leading to saturation.

Together, these conditions ensure that all goods entering the system can eventually be processed and removed, enabling feasible and stable operating conditions across time.

To determine which states must be known in the initial condition, it is necessary to find all states in the state equation that depend on the state at cycle $k-1$. These represent the minimal set of states required to ensure a valid state evolution. An equivalent approach is to inspect the matrix C: each column containing a finite non-zero entry indicates that the corresponding state (which is the row in the state vector) must be initialised.

Using this approach, the following states must be included in the initial state x_0:

$$d_{41}(0), \quad d_{42}(0), \quad d_{43}(0), \quad \rho_{42,\text{real}}(0), \quad \rho_{43,\text{real}}(0), \quad s_1(0), \quad s_2(0), \quad s_3(0)$$

All system states can be uniquely determined if these 8 values are known, along with the specific system parameters.

8-3-2 Growth Rate, Fixed Points, and Periodicity Analysis

Now that the system is fully defined, the growth rate, fixed points and period lengths can be determined. This system has around 6 sextillion ($6 \cdot 10^{21}$) different footprint matrix pairs. This is an enormous amount which is impossible to analyse using the LPP strategy from [5]. Remember that during modelling, it was assumed that the loading and unloading rates were unknown, and one would not know which would be the bottleneck. This introduced a lot of extra possible affine terms to the system, which has now resulted in a lot of unreachable modes and extra footprint matrix pairs to check once the system parameters have been chosen. Note that modes here are as defined by Definition 5.4 and do not refer to the 3 interaction modes described in Subsection 8-2-1. When removing all these unreachable terms, the size of the system can be reduced significantly, to around 9 trillion ($9 \cdot 10^{12}$) different footprint matrix pairs. Even if one tries to solve this using the LPP method at 0.1 seconds per LPP, it would still take approximately 28 years to evaluate every footprint matrix pair. Which is why the proposed MILP strategy from Chapter 5 will be applied. Furthermore, the periodicity of the system will also be evaluated using the extended periodic ABCD canonical form from Chapter 6 for $p = 1$ and $p = 2$. The analysis was performed in MATLAB using Mosek as the solver.

Analysis of Dominant Modes for p = 1

After prepossessing using the MILP, it was found that out of all the finite entries in the rows with more than one finite entry, 24 were present in at least one dominant mode. This reduced the number of potentially dominant modes from around 9 trillion to 4096. After running the entire MILP and exploring the entire tree, 1024 footprint matrix pairs are found to yield a growth rate. This analysis took around 2.5 hours. All 1024 footprint matrix pairs yield a growth rate of $\lambda^* = 11.75$. This eigenvalue can somewhat be expected as it must be at least more than the maximal round trip travel time, which is it, as this is given by 8. All 1024 footprint matrix pairs G_{A_θ} and G_{B_θ} yield an eigenvector. After accounting for numerical rounding with a tolerance of 10^{-5}, only 3 unique eigenvectors remain. Which will be denoted by $V = \{v_1, v_2, v_3\}$ where $v_i = (x_{e,i}^\top, y_{e,i}^\top, w_{e,i}^\top)$.

Now that only the unique eigenvectors remain, it will be checked whether they are truly eigenvectors. By verifying whether the following holds for all eigenvectors:

$$x_e + s \cdot \lambda^* = A \otimes (B \otimes' (C \cdot x_e + D \cdot (x_e + s \cdot \lambda^*))) \tag{8-73}$$

This is true for all found eigenvector eigenvalue pairs, so all eigenvector eigenvalue pairs are confirmed to be true eigenvalues and eigenvectors. However, all found solutions might not be all solutions. As the constraints can have more invariant directions where eigenvectors can lie. By substituting the eigenvalue λ^* into the equality and inequality constraints of the LPP in Subsection 3-3-2, the following will be obtained:

$$H_{eq} \cdot v = h_{eq}, \quad H_{ineq} \cdot v \leq h_{ineq} \tag{8-74}$$

The true matrices are too large to display here, however their sizes are; $H_{eq} \in \mathbb{R}^{199\times199}$, $h_{eq} \in \mathbb{R}^{199\times1}$, $H_{\text{ineq}} \in \mathbb{R}^{40\times199}$ and $h_{\text{ineq}} \in \mathbb{R}^{40\times1}$. The set of fixed-points can be described by $\mathcal{V}_{\lambda^*} = \{v \mid H_{\text{ineq}} \cdot v \leq h_{\text{ineq}}\}$[5] where $v = v^* + \sigma_1 g_1 + \sigma_2 g_2 + \cdots + \sigma_f g_f$ as per Subsection 3-3-2 where the number of different terms of g_i is related to the rank deficiency of H_{eq}. For exactly half of them, which is 512, the rank deficiency is equal to 1, while for the other half, the rank deficiency is equal to 1. This means that there are 1 to 2 direction vectors to describe all fixed points for this system. Thus, the rank of the matrix $V_e = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$ must be a maximum of 2, where each column of V_e is an eigenvector. However, after investigation, one of the 3 eigenvectors is simply the time-shifted version of another vector. This means that V_e is actually just $\{v_1, v_2\}$. The rank of V_e is equal to 2. So the solution to the MILPs has yielded all existing direction vectors. The last vector could be constructed from the matrix H_{eq}. Finally, the eigenvector for the growth rate $\lambda = 11.75$ can be described by:

$$v = v^* + \sigma_1 \cdot g_1 + \sigma_2 \cdot g_2 \tag{8-75}$$

Since the system is time invariant, the first scaling factor σ_1 is always unbounded in both directions [5]. The direction vectors g_1 and g_2 can be determined quite easily. Since v consists of x, y, w, it is important to make sure g_i is of the same size. The first invariant direction g_1 is given by $s = \begin{bmatrix} \mathbf{1}_{n_t}^\top & \mathbf{0}_{n_q}^\top \end{bmatrix}^\top$ which in terms of g leads to:

$$g_1 = \begin{bmatrix} g_{1,x} \\ g_{1,y} \\ g_{1,w} \end{bmatrix} \tag{8-76}$$

where $g_{1,w} = (C + D) \cdot s$, $g_{1,y} = B \otimes' ((C + D) \cdot s)$ and $g_{1,x} = s$ g_1 and g_2 are too large to show here. The second direction vector, g_2, can be found when examining the found eigenvectors

v_1 and v_2. The inflow into the system was chosen such that truck 1 is always full and there is no accumulation of goods in stack $s_1(k)$. This makes the stack size $s_1(k)$ the second direction vector. In other words, $s_1(k)$ is completely decoupled from the system dynamics. This means that $g_{2,x}$ is a zero vector with a single 1 at the location of state $s_1(k)$, which is entry $x(41)$. Then $g_{2,w} = (C + D) \cdot g_{2,x}$, $g_{2,y} = B \otimes' ((C + D) \cdot g_{2,x})$. This also results in the bound for σ_2 being; $0 \leq \sigma_2 < \infty$.

It might look strange that half of the H_{eq} matrices have a rank deficiency of 1, while the other half have a deficiency of 2. Normally, the expectation is that all H_{eq} matrices share the same rank deficiency. This difference, however, has an explanation.

In the cases where the deficiency is 1, the dominant mode of the system makes the departure time of truck 1 at node 1, (8-56), picks the second affine term and taking the entire stack with it. For that to happen, the stack size must stay below the capacity of truck 1.

But the parameters were chosen so that the inflow matches the capacity exactly over one cycle. That means it makes no difference in (8-56) whether the first or second affine term is taken. For Heq, though, this choice does matter. If the first term is taken, the other direction vector appears, but if the second term is taken, it doesn't. However, in both cases, the direction vector corresponding to the stack size $s_1(k)$ is a valid shift-invariant direction.

Analysis of Dominant Modes for p = 2

When analysing the system in extended periodic ABCD form, the number of possible footprint matrix pairs increases drastically. Specifically, the number of pairs scales quadratically with the period p under consideration. For example, what was previously $9 \cdot 10^{12}$ possible pairs for $p = 1$, becomes $81 \cdot 10^{24}$ for $p = 2$.

In contrast, the preprocessing step scales linearly with the size of the system, not the period. This makes it far more efficient for pruning the search space. In the previous case, the preprocessing step required 71 MILP calls. For $p = 2$, this increased to 142 calls. While each MILP instance grows in size (and therefore runtime), the preprocessing remains tractable. In total, preprocessing for $p = 2$ took around 1 hour, and reduced the number of candidate pairs from $81 \cdot 10^{24}$ down to approximately $3.5 \cdot 10^{11}$.

Although this is a significant reduction, it is still not enough. Exploring the entire tree of $3.5 \cdot 10^{11}$ possible valid pairs would still be computationally infeasible, as with 0.1 seconds per call, it would take around 100 years. Therefore, a manual pruning step was used based on insights from the $p = 1$ case and knowledge of the system.

From prior analysis, it is known that when both truck 2 and truck 3 arrive at the same time, no periodic behaviour exists. However, all such pairs remain in the current search space. Additionally, for $p = 2$, we are only interested in periodic orbits of length at most 2. Since the starting point of the orbit is arbitrary due to the ability to permute the block ordering of G_A, G_B, and x_e, this introduces redundant twin solutions and unnecessarily inflates the search space.

To eliminate these redundant cases, we manually constrain the MILP by setting specific integer variables in p and q to zero. This targeted reduction brings the number of valid footprint matrix pairs down to just 576.

It is important to note that:

- Any solution found for $p = 1$ remains valid here, as discussed in Chapter 6.
- Any detected periodic orbit for $p = 2$ actually appears twice in the search space, once per possible starting point. Meaning twin solutions exist, and one is always omitted due to the manual pruning.

After running the entire MILP and exploring the reduced tree, 9 footprint matrix pairs are found to yield a growth rate. The total analysis took just over 2 hours in total. All 9 footprint matrix pairs yield a growth rate of $\hat{\lambda} = 23.5$. This eigenvalue can somewhat be expected as the system is twice the size of the original system with $p = 1$, which had an eigenvalue of $\lambda^* = 11.75$. All 9 solutions represent a periodic orbit; this means that the permuted block ordering of G_A, G_B, and x_e must also be a valid footprint matrix for a fixed point. Additionally, the pairs found for $p = 1$ would also have been valid, but since the system is twice the size, all permutations of pairs would also be valid, which yields a further 1024^2 pairs. The same analysis can be done for the case of $p = 2$ as was done for $p = 1$; now only the periodic solutions found here will be considered, since they have a real difference with respect to the solutions found in Section 8-3-2. Also, note that the twin pairs of the found eigenvectors will not be considered. The set of found eigenvectors will be denoted by $V = \{v_1, v_2, \ldots, v_9\}$ where $v_i = (x_{e,i}^\top, y_{e,i}^\top, w_{e,i}^\top)$.

All eigenvectors are verified whether they are truly eigenvectors. By verifying whether the following is true for all eigenvectors:

$$x_e + s \cdot \lambda^* = A \otimes (B \otimes' (C \cdot x_e + D \cdot (x_e + s \cdot \lambda^*))) \tag{8-77}$$

This is true for all found eigenvector eigenvalue pairs, so all eigenvector eigenvalue pairs are confirmed to be true eigenvalues and eigenvectors. However, all found solutions might not be all solutions. As the constraints can have more invariant directions where eigenvectors can lie. By substituting the eigenvalue λ^* into the equality and inequality constraints of the LPP in Subsection 3-3-2, the following will be obtained:

$$H_{eq} \cdot v = h_{eq}, \quad H_{ineq} \cdot v \leq h_{ineq} \tag{8-78}$$

The true matrices are too large to display here, however their sizes are; $H_{eq} \in \mathbb{R}^{398\times398}$, $h_{eq} \in \mathbb{R}^{398\times1}$, $H_{\text{ineq}} \in \mathbb{R}^{80\times398}$ and $h_{\text{ineq}} \in \mathbb{R}^{80\times1}$. Notice that this is exactly twice as large as the H, h matrices found in Section 8-3-2. The set of fixed-points can be described by $\mathcal{V}_{\lambda^*} = \{v \mid H_{\text{ineq}} \cdot v \leq h_{\text{ineq}}\}$[5] where $v = v^* + \sigma_1 g_1 + \sigma_2 g_2 + \cdots + \sigma_f g_f$ as per Subsection 3-3-2. Where the number of different terms is related to the rank deficiency of H_{eq}. For all of them, the rank deficiency is equal to 3. This means that there are 3 direction vectors to describe all fixed points for this system. Thus, the rank of the matrix $V_e = \begin{bmatrix} v_1 & v_2 & \ldots & v_9 \end{bmatrix}$ must be a maximum of 3, where each column of V_e is an eigenvector. However, after accounting for numerical rounding with a tolerance of 10^{-5}, only 4 unique eigenvectors remain, where one was simply another vector time-shifted. This means that V_e is actually just $\{v_1, v_2, v_3\}$. The rank of V_e is equal to 3. So the solution to the LPP has yielded all existing directions. This means that an eigenvector can be described by:

$$v = v^* + \sigma_1 \cdot g_1 + \sigma_2 \cdot g_2 + \sigma_3 \cdot g_3 \tag{8-79}$$

Since the system is time invariant, the first scaling factor σ_1 is always unbounded in both directions. The direction vectors g_1 and g_2 can be determined quite easily. Since v is made up of x, y, w, it is important to make sure g_i is of the same size. The first invariant direction is given by $s = \begin{bmatrix} \mathbf{1}_{n_t}^\top & \mathbf{0}_{n_q}^\top \end{bmatrix}^\top$ Which in terms of g leads to:

$$g_1 = \begin{bmatrix} g_{1,x} \\ g_{1,y} \\ g_{1,w} \end{bmatrix} \tag{8-80}$$

where $g_{1,w} = (C + D) \cdot s$, $g_{1,y} = B \otimes' ((C + D) \cdot s)$ and $g_{1,x} = s$ g_1 and g_2 are too large to show here. g_2 can also be found when examining the found eigenvectors. The inflow into the system was chosen such that truck 1 is always full and there is no accumulation of goods in stack $s_1(k)$. The stack size $s_1(k)$ is the second direction vector. This means that $g_{2,x}$ is a zero vector with a single 1 at the location of state $s_2(k)$, which is state 41 and 92. Then $g_{2,w} = (C + D) \cdot g_{2,x}$, $g_{2,y} = B \otimes' ((C + D) \cdot g_{2,x})$. This also results in the bound for σ_2 being; $0 \leq \sigma_2 < \infty$. The other shift invariant direction is a direction which corresponds to the behaviour where the time difference between truck 2 and 3 arriving at node 4 is a lot smaller; this also has the effect that the truck loads are more similar. This was found by analysing the found eigenvectors.

8-3-3 System Simulations for Periodic Behaviour

In the previous Section, the growth-rate fixed points and period of the system were analysed and identified. It was found that the system has a periodic region with a period equal to 1 and a periodic region with a period equal to 2. In this Section, the system will be simulated, where the aim is to obtain uniform behaviour. This means that the stop time at a node will be the same over every cycle, as well as the quantity states remaining constant over every cycle. This corresponds to the number of goods entering the system being equal to the number of goods leaving it. The eigenvector of the system will be taken to simulate the system. First, the system with a period of 1 is simulated and then the system with a period of 2. For simulating the system with a period of 2, one of the periodic points will be taken as the starting point.

Simulation for Period p = 1

Using the fixed point v_1 found in Subsection 8-3-2, the system will be simulated for $k = 10$. The simulation yields Figure 8-3, 8-4, 8-5 and 8-6. What is clearly visible in the figures is that all quantity states are constant over the event cycles with $\rho_1(k) = 75$, $\rho_{42}, \rho_{43} = 37.5$, $s_1(k) = 50$ and $s_2(k), s_3(k) = 0$. The arrival times and departure times are all parallel with a uniform growth rate of $\lambda^* = 11.75$. This indicates a uniform timetable for when trucks will arrive and depart. Figure 8-6 shows that truck 1 is always the first one to arrive at node 4, while trucks 2 and 3 arrive simultaneously at a later time, and all trucks leave simultaneously as well. This indicates that the system with these chosen parameters is in mode 1.

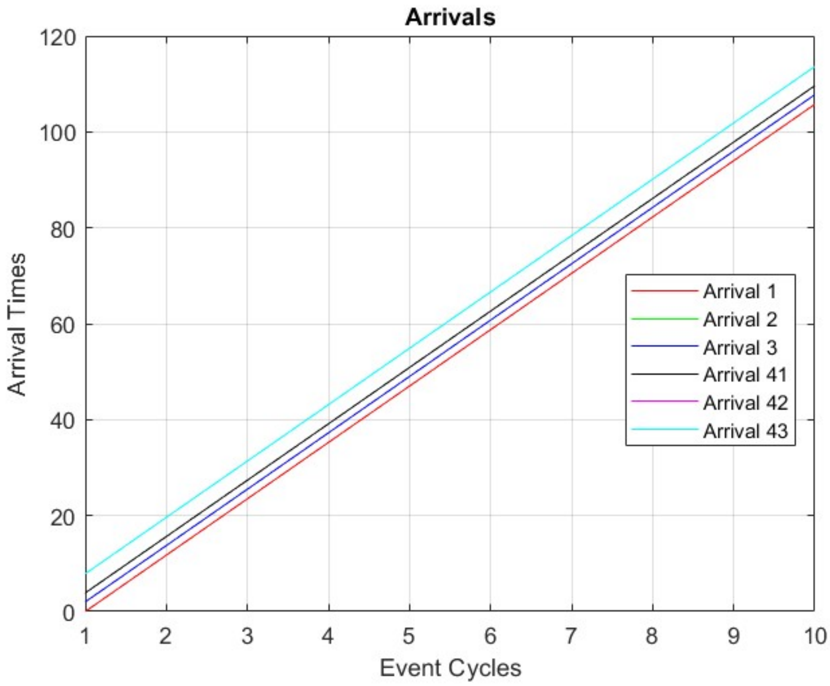


Figure 8-3: Truck arrival times of the uniform transportation network with 10 cycles with a period of 1

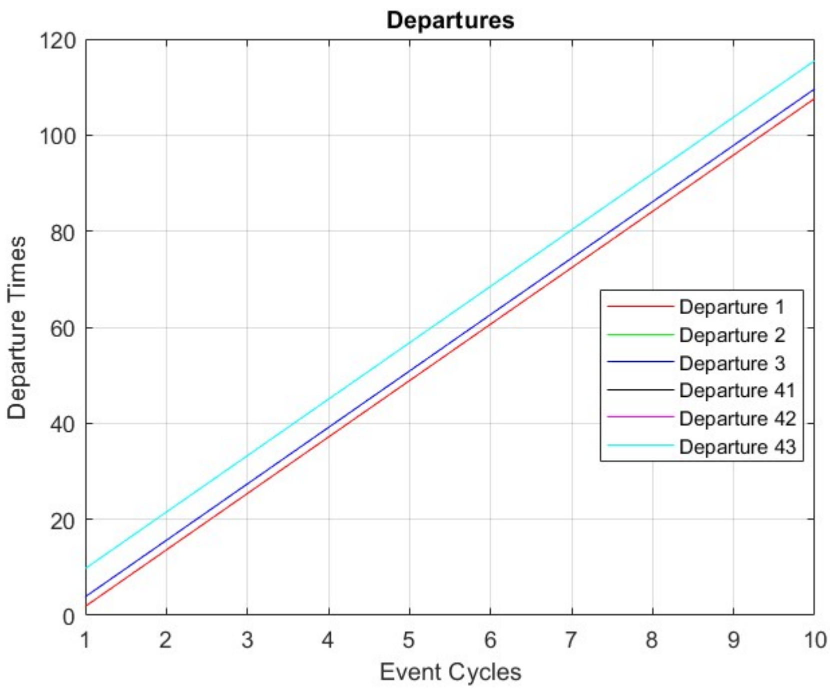


Figure 8-4: Truck departure times of the uniform transportation network with 10 cycles with a period of 1

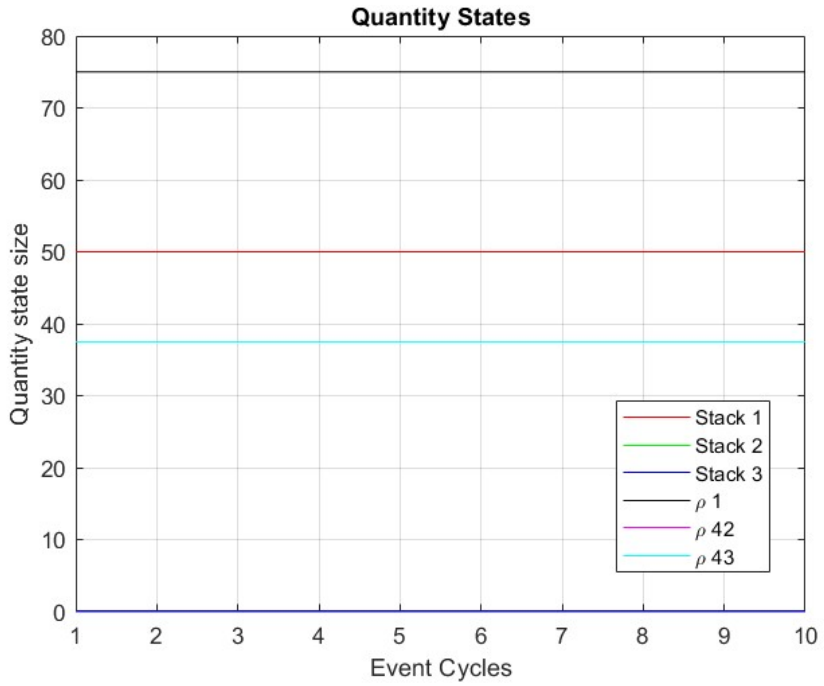


Figure 8-5: Quantity states of the uniform transportation network with 10 states with a period of 1

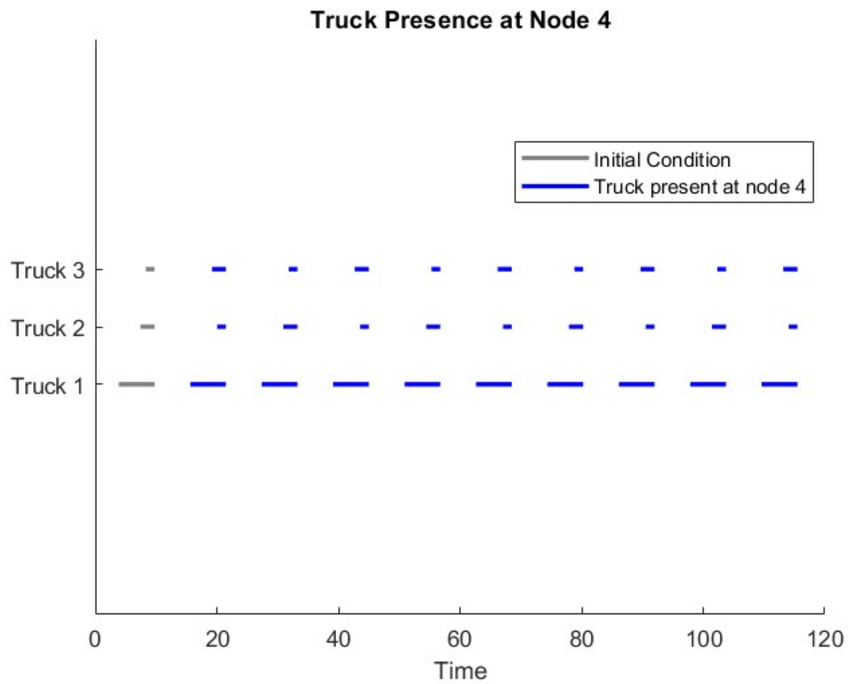


Figure 8-6: Trucks present at node 4 of the periodic transportation network with 10 states with a period of 2

Simulation for Period p = 2

Using the fixed point v_1 found in Section 8-3-2, the system will be simulated for $k = 10$, but now for $p = 2$. The simulation yields Figure 8-7, 8-8, 8-9 and 8-10. What is clearly visible in the figures is that the quantity states $s_1(k)$, $s_2(k)$, $s_3(k)$ and $\rho_1(k)$ are constant over the event cycles with $\rho_1(k) = 75$ $s_1(k) = 50$ and $s_2(k), s_3(k) = 0$. However, the quantity states $\rho_{42}(k)$ and $\rho_{43}(k)$ oscillate during each event cycle, switching values with each other while remaining bounded between 28.75 and 46.25. The arrival and departure times of the trucks follow parallel trajectories with a uniform average growth rate of $\lambda^* = 11.75$, except for the arrival times at node 4 of trucks 2 and 3. These arrival times alternate between growth rates of 10.875 and 12.625 on successive cycles, averaging out to the same $\lambda^* = 11.75$. A similar pattern occurs for the departure times of trucks 2 and 3 at nodes 2 and 3, respectively. These shifts are in phase with each other but out of phase relative to the truck indices. This behaviour is a bit hard to see in Figure 8-7 and 8-8, which is why Figure 8-11 and 8-12 are zoomed-in versions, where this behaviour is more clearly visible.

Despite the oscillations, the system still resembles a consistent timetable in which truck arrivals and departures follow a predictable pattern, though not identical in every cycle. As shown in Figure 8-6, truck 1 consistently arrives first at node 4, while trucks 2 and 3 arrive slightly later, alternating their order. All trucks then depart simultaneously. This behaviour indicates that, for the given parameters, the system operates in mode 1.

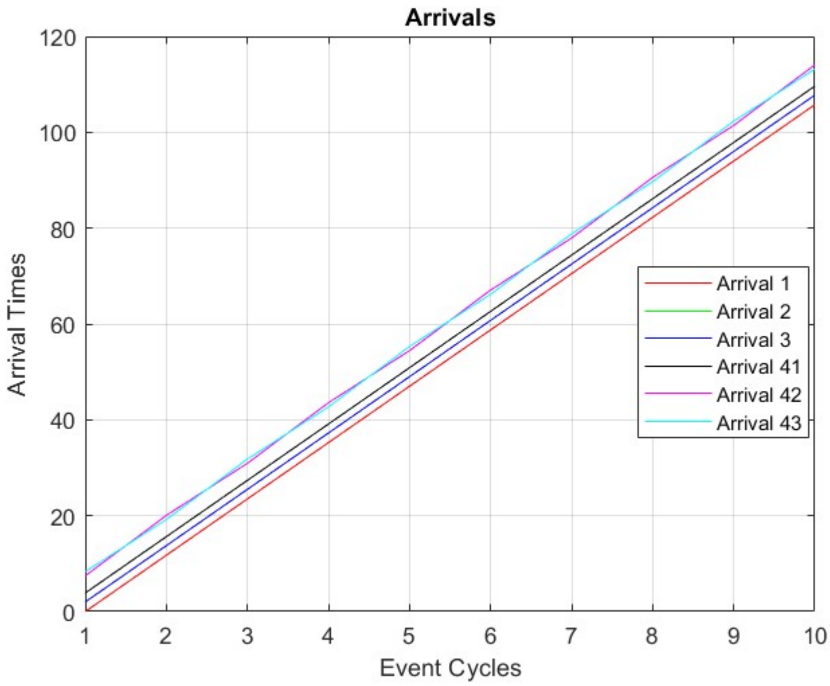


Figure 8-7: Truck arrival times of the periodic transportation network with 10 cycles with a period of 2

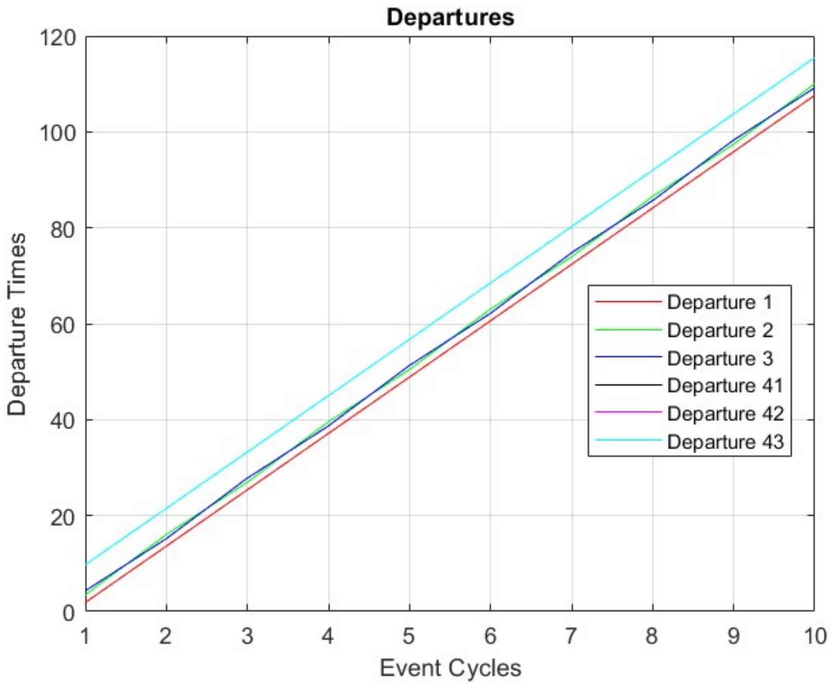


Figure 8-8: Truck departure times of the periodic transportation network with 10 cycles with a period of 2

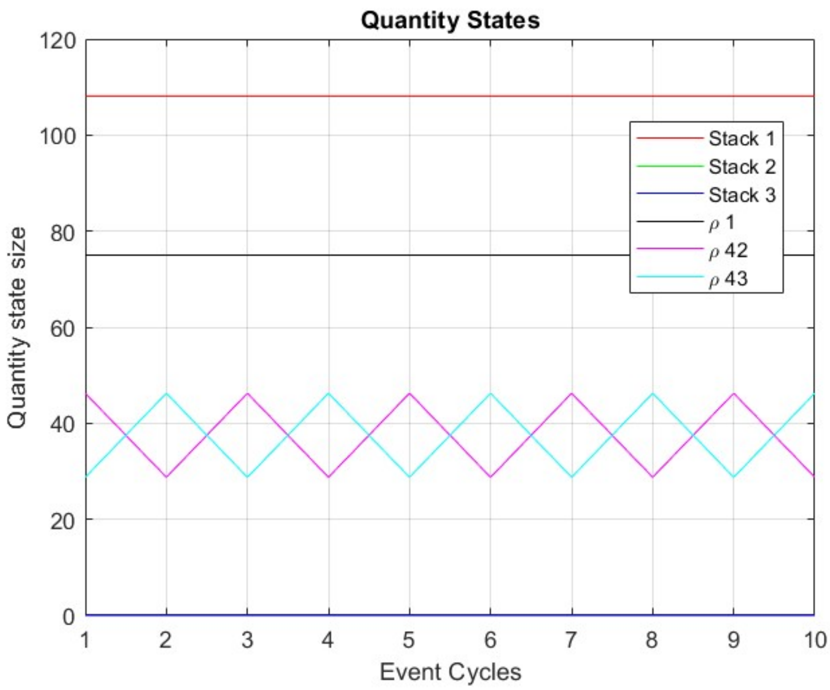


Figure 8-9: Quantity states of the periodic transportation network with 10 states with a period of 2

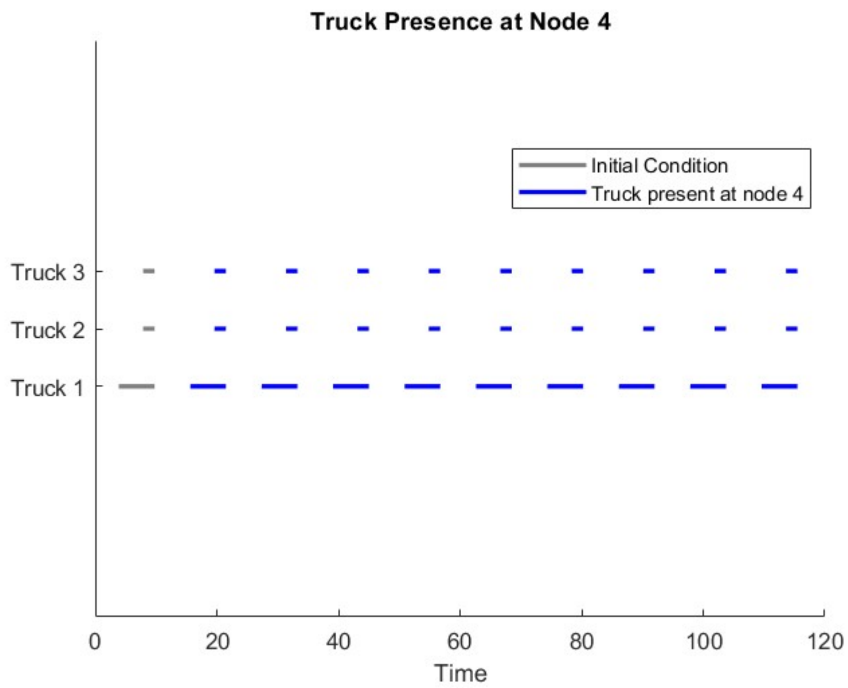


Figure 8-10: Trucks present at node 4 of the uniform transportation network with 10 states with a period of 2

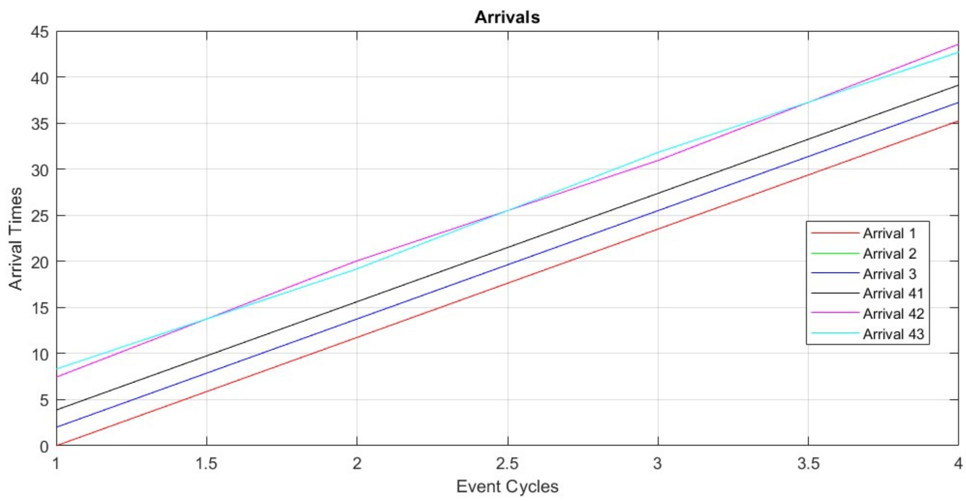


Figure 8-11: Zoomed in truck arrival times of the periodic transportation network

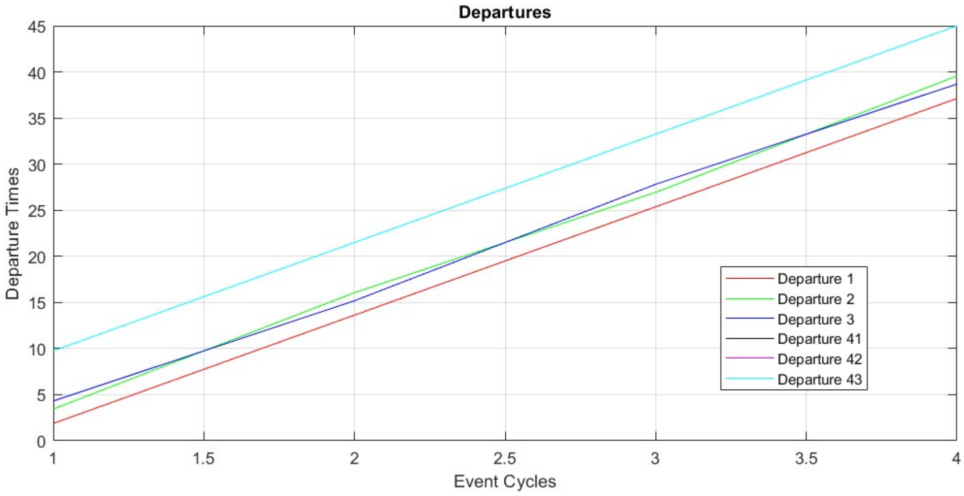


Figure 8-12: Zoomed in truck departure times of the periodic transportation network

8-3-4 Stability of the Transportation System

The concept of stability for MMPS systems was extensively discussed in Chapter 4. Bounded buffer stability in terms of MMPS systems refers to the boundedness of the difference between the time states at each event k. According to the definition for DE systems, a DE is bounded buffer stable if these buffer levels remain constant over time. In Subsection 8-3-3, the transportation system was simulated, and the growth rates, fixed points and periodic orbits were identified. This section will investigate whether these points and orbits are bounded-buffer stable or not. To conclusively determine whether an implicit MMPS system with a given growth rate λ is bounded-buffer stable; the system must first be normalised and then linearised. The normalisation procedure follows the same process as in Subsection 3-3-2. The resulting normalised form of an implicit MMPS system is given by the following expression:

$$\tilde{x}(k) = \tilde{A} \otimes \left(\tilde{B} \otimes' (C \cdot \tilde{x}(k-1) + D \cdot \tilde{x}(k))\right) \tag{8-81}$$

Once the system has been normalised, the system can be linearised using the theory provided by Section 4-3. This definition will be shown again below:

Definition 8.1. *(Linearisation of an MMPS system [5])*
Any normalised MMPS system can be transformed into a linear representation in conventional algebra for all $\tilde{x}(k) \in \Omega_\theta,\ k \in \mathbb{Z}^+$ as follows:

$$\begin{aligned} \tilde{x}_\theta(k) &= M_\theta \cdot \tilde{x}_\theta(k-1) \\ M_\theta &= (I - M_1)^{-1} \cdot M_2 \\ M_1 &= G_{\mathrm{A}_\theta} \cdot G_{\mathrm{B}_\theta} \cdot D \\ M_2 &= G_{\mathrm{A}_\theta} \cdot G_{\mathrm{B}_\theta} \cdot C \end{aligned} \tag{8-82}$$

if the inverse $(I - M)$ exist, where Ω_θ is a polyhedron wherein the linearisation is valid

Stability of the Fixed Points with p = 1

Simulating the system initialised in the found fixed point yields a constant growth in the time states and constant quantity states. This is an indication that the system is stable for this fixed point. However, it is only possible to know for sure by checking the bounded buffer stability for the linearised systems. All 1024 fixed points with dominant modes are checked to see whether they are bounded buffer stable. This means that for all found dominant modes, the system will be normalised and linearised. When performing these steps and checking the eigenvalues, the multiplicity of the eigenvalues equal to 1, and the Jordan blocks, the following results were obtained:

- All 1024 linearizations are bounded-buffer stable.
- 512 M_θ matrices have 2 multiplicative eigenvalues equal to 1.
- 512 M_θ matrices have 1 multiplicative eigenvalue equal to 1.
- None of the the M_θ matrices have eigenvalues larger than 1.

So all found fixed points and accompanying footprint matrices G_{A_θ} and G_{B_θ} yield a bounded buffer stable system. This large number can be attributed to some redundancy inside the system, resulting in a large number of footprint matrix pairs yielding the same result. The 1024 unique footprint matrix combinations are obtained by a total of 65 unique G_{A_θ} matrices and 17 unique G_{B_θ} matrices; they all yield a growth rate of $\lambda = 11.25$.

Stability of the Periodic Orbits with p = 2

The simulated system with a periodic orbit also appears to be stable; however, this must be verified by checking the bounded buffer stability of each linearization. All 9 periodic orbits with dominant modes are checked to see whether they are bounded buffer stable. This means that for all found dominant modes, the system will be normalised and linearised. Again, the twin pairs will not be checked as this method validates the entire periodic orbit as a whole, and so it is not needed to check the twins. Also, only the periodic orbit will be checked and not the semi-dominant modes, as the entire orbit is of interest, and not whatever happens in between. When performing these steps and checking the eigenvalues, the multiplicity of the eigenvalues equal to 1, and the Jordan blocks, the following results were obtained:

- All 9 linearisations of the periodic orbit are bounded buffer stable.
- All M_θ matrices have 3 multiplicative eigenvalues equal to 1.
- None of the the M_θ matrices have eigenvalues larger than 1.

The 9 unique footprint matrix combinations are obtained by a total of 1 unique G_{A_θ} matrix and 9 unique G_{B_θ} matrices; however, they all yield a growth rate of $\lambda = 11.25$. But remember that the twins are not considered, so this number is, in actuality, at least double.

8-3-5 Maximal Invariant Set of the Transportation System

In the previous section, the bounded-buffer stability of both the fixed points and periodic orbits was investigated. All found fixed points and periodic orbits were found to be bounded buffer stable. In this section, the invariant set for each linearised system is determined, as it is interesting to know whether the system will stay in a specific linearization region or not. This region is called the maximal invariant set and was discussed in Chapter 4. This set can be obtained by running Algorithm 2, where $\text{Pre}\,(\Omega_\theta) = \{x \in \mathbb{R}^n \mid H \cdot M \cdot x \leq h\}$[5]. The algorithm is presented again below;

Algorithm 6 Maximal positive invariant set [5]

Input: M_θ, Ω_θ
Output: $\mathcal{O}_\infty$
$\mathcal{O}_0 \leftarrow \Omega_\theta, k \leftarrow -1$
Repeat
$k \leftarrow k + 1$
$\mathcal{O}_{k+1} \leftarrow \text{Pre}\,(\mathcal{O}_k) \cap \mathcal{O}_k$
Until: $\mathcal{O}_{k+1} = \mathcal{O}_k$
$\mathcal{O}_\infty \leftarrow \Omega_k$

Maximal Invariant Set of Fixed Points p=1

To assess the bounded-buffer stability of the fixed points, the maximal invariant set was approximated for each of the 1024 linearised systems using Algorithm 6. Each system, defined by a fixed point and its associated matrix M_θ and region $\Omega_\theta = \{x \in \mathbb{R}^n \mid H \cdot x \leq h\}$, was tested for invariance. The algorithm was allowed to run for a maximum of 50 iterations per system. This number is arbitrarily chosen, but will turn out to be large enough. Due to the large number of linearizations, 1024, it is not worthwhile to visualise all results in a large table.

Out of the 1024 cases, all terminated within a maximum of 16 iterations. However the some resulted in empty invariant sets. There are 312 linearisations which resulted in a non-empty invariant set, while the remaining 712 converged to an empty set. Most did terminate in much less time, only needing 1-5 iterations. Having stable linearisations with an empty invariant set. This is most likely attributable to the linearisation being mathematically stable due to the redundancy; however, not sustainable during state evolutions.

An important observation is that there is no direct link between the multiplicity of eigenvalues equal to one and the existence of a non-empty invariant set.

Maximal Invariant Set of Periodic Orbits p=2

For periodic orbits of period $p = 2$, the maximal invariant set can be approximated in the same way as for fixed points. Each periodic orbit gives rise to a linearised system that can be analysed using Algorithm 6, with the goal of evaluating bounded-buffer stability and identifying invariant regions in the state space.

A total of 9 periodic orbits were identified through simulation and linearisation procedures, and each corresponding linearised system was tested for invariance using the same setup: a maximum of 50 iterations and the region defined by $\Omega_\theta = \{x \in \mathbb{R}^n \mid H \cdot x \leq h\}$. In contrast to the fixed-point case, all 9 periodic systems yielded non-empty invariant sets, confirming their stability under the given dynamics.

The invariant sets were found with the number of iterations, ranging from just 1 to a maximum of 9 iterations. This shows that the algorithm converged rapidly for all periodic orbits and suggests a high degree of regularity in their dynamic behaviour.

The results are summarized in Table 8-2, where for each orbit the following is listed: the number of eigenvalues equal to 1, whether the system is bounded-buffer stable, the rank deficiency of H_{eq}, whether an invariant set was found, how many iterations it took, and whether the final set was empty.

These findings highlight that, unlike the fixed-point case, all periodic linearizations resulted in stable behaviour with non-empty invariant regions.

#	Multiplicity e.v. $1 M_\theta$	BB stable?	H_{eq} rank def.	$\mathcal{O}_\infty$ found	Iter	Empty
1	3	Yes	3	Yes	1	No
2	3	Yes	3	Yes	1	No
3	3	Yes	3	Yes	2	No
4	3	Yes	3	Yes	1	No
5	3	Yes	3	Yes	1	No
6	3	Yes	3	Yes	9	No
7	3	Yes	3	Yes	2	No
8	3	Yes	3	Yes	2	No
9	3	Yes	3	Yes	1	No

Table 8-2: Analysis of the stability of 9 periodic orbits of the transportation system

Chapter 9

Conclusions and Contributions

In this Chapter, the research carried out in this thesis is concluded. A reflection on the posed research questions from Chapter 1 is given. Each main research question with its accompanying sub-questions will be discussed in a dedicated section. This means that Section 9-1 reflects on research question 1. Section 9-2 will reflect on research question 2, and Section 9-3 on research question 3. Finishing off the chapter is Section 9-4, where an overview of all concrete academic contributions is given.

9-1 On Scalable Analysis

This section addresses the first research question and its sub-questions by reflecting on the results presented in Chapter 5. The research question and its sub-questions are stated as follows:

- How can a hybrid approach combining search trees, LPP, and MILP reduce the computational complexity of analysing Max-Min-Plus-Scaling Systems?
  - (a) How can the existing explicit MILP algorithm be extended to also apply to general implicit MMPS systems?
  - (b) How can a search tree be used to systematically explore and prune the search for eigenvalues of MMPS systems to avoid redundant or infeasible paths?

For the first sub-question, the existing MILP algorithm for explicit MMPS systems can be extended to apply to general implicit MMPS systems. By drawing inspiration from the normalisation of implicit MMPS systems, and the derivation of the original MILP, it was possible to extend the MILP to also find an eigenvalue and eigenvector of an implicit MMPS system. Answering the first sub-question. However, as is known from general MMPS systems, they can have multiple eigenvalues and eigenvectors. This means that in the search space, there can be more than one feasible solution. As the goal is to find all feasible solutions, the second sub-question comes into play. By externalising a depth-first search method, all feasible solutions can be found. With infeasible systems, the infeasible is often quickly detected, meaning that a search into an infeasible direction is caught quickly. Due to this infeasibility occurring somewhere in the search space, which can be represented by a tree, it also allows for early pruning in the infeasible directions, greatly reducing the computational complexity if there are only a handful of feasible solutions.

Concluding the first research question, it is possible to greatly reduce the computational complexity of analysing growth rates and fixed points in MMPS systems by applying an MILP algorithm with a depth-first search strategy, together with a prepossessing step that yields a large reduction in the search space. If the reduction due to the preprocessing is not large enough, then switching the solution to solving according to the regular LPP strategy on the reduced search space still leads to a significant reduction in computational power required.

9-2 On Periodicity

In this section, the second research question is addressed. With the accompanying sub-questions. This research was performed in Chapter 6. Let us first recall the research question:

How can new theoretical insights into the structure and dynamics of MMPS systems contribute to more effective analysis of periodic system behaviour?

(a) How can periodic MMPS systems be transformed to allow for periodicity and stability analysis?

(b) How can the stability of periodic orbits be guaranteed?

The introduction of the extended periodic ABCD form makes it possible to represent any periodic MMPS system as an equivalent extended MMPS system with a period of one. With the period reduced to one, the existing analysis methods for MMPS systems can be applied directly, including the MILP algorithm introduced in Chapter 5.

In practice, the period of a system is not always known in advance. For max-plus and min-plus systems, the maximum possible period length is known, which allows for a recursive search using the extended periodic ABCD form until this bound is reached. For general MMPS systems, however, such a bound has not yet been determined.

Because the extended periodic ABCD form captures the entire periodic orbit, existing stability criteria can be used to assess and guarantee the stability of the full orbit. Notably, only a subset of the extended system needs to be considered for this purpose, as analysis of its structure shows that stability depends on only part of the system.

9-3 On Modelling

In this Section, the third research question is addressed, with the accompanying sub-questions. This research was performed in Chapter 7 and 8. Let us first recall the research question:

Is it possible to model, simulate and analyse a transportation network such that it closely resembles reality?

(a) Can the system equations be written in such a way as to incorporate all different arrival and departure patterns?

(b) What insights can be obtained from analysing the dynamical behaviour of a 4-node transportation network?

(c) How can the system be generalised to allow for more complex modelling, simulation and analysis?

(d) Can a framework be created to allow for easy implementation of a generalised transportation network?

The case study revealed that the 4-node system was challenging to model due to its six distinct operating modes, determined by arrival times, capacities, and loading/unloading speeds. To address this, a method was developed for expressing a switching MMPS system as a single MMPS system description, enabling the modelling of these complex interactions.

Some operating modes overlapped, but with careful derivations, these overlapping regions were removed, allowing the system to be analysed effectively. The resulting model served as a strong test case for applying the theories developed in both Chapter 5 and Chapter 6. The MILP approach proved effective, and periodic behaviour was successfully identified. This demonstrated that the MMPS framework can be applied beyond the URS.

The work also showed that a more general modelling framework is possible. By introducing MMPS subsystems and deriving time-invariance and solvability conditions for them, the modelling process can be parallelised. Additionally, a node-based framework was created, allowing users to describe a system at a high level and automatically obtain the full set of system equations. This is achieved using a database and a structure similar to the URS to construct the system matrices.

In conclusion, this case study demonstrates that it is possible to model, simulate, and analyse a transport network in a way that closely resembles real-world operations. Including an extension to allow for even more general transportation systems to be modelled as MMPS systems.

9-4 Contributions

This thesis contributed to the research in the field of Systems and Control and Discrete Event Systems, specifically to Max-Min-Plus-Scaling system through the following results:

- Developed an MILP formulation for identifying growth rates and fixed points in implicit MMPS systems.
- Developed a search algorithm for applying the MILP algorithm to implicit MMPS systems.
- Developed a Recursive standalone program to analyse general MMPS systems much more efficiently than the current state of the art.

- Greatly reduced the computational load for analysing MMPS systems.
- Proposed an extended periodic ABCD for for periodic MMPS systems with a period large than 1.
- Derived stability criteria for periodic orbit of MMPS systems.
- Proposed MMPS sub-systems description for transportation systems as well as for general MMPS systems.
- Proposed solvability and time invariance conditions for both open-loop and closed loop MMPS sub-system integrations.
- Proposed a method of writing switching MMPS systems into a single system description.
- Developed a framework for modelling transportation networks using individual nodes.
- Developed a standalone program to transform a high-level transportation system description into a full MMPS system of equations in ABCD form.
- Derived a complex 4-node transportation system.
- Analysed the 4-node transportation system using theoretical results regarding the MILP algorithm and periodicity.

Chapter 10

Recommendations for Future Work

The research presented in this thesis provides a solid foundation for the analysis, modelling, and simulation of MMPS systems, but there are still many open directions worth exploring. Some relate to deepening the theoretical understanding of these systems, while others aim to extend the modelling framework and make it more practical for real-world applications. The following list outlines several promising directions for future work.

- **Derive conditions for monotonicity and non-expansiveness in implicit MMPS systems**

For explicit MMPS systems, the conditions for monotonicity and non-expansiveness are already well established. Extending these results to implicit MMPS systems could reveal a new subclass of topical implicit systems. If such conditions are found, the eigenvalue search described in Chapter 5 would no longer be necessary for these cases, as a single growth rate would be guaranteed. In practice, this would allow the MILP formulation to be used as the sole analysis step, significantly simplifying computations.

- **Reduce the search space for large or extended MMPS systems**

The current combination of MILP and preprocessing steps greatly reduces the search space for growth rates and fixed points. However, many infeasible paths remain after preprocessing, and these are still explored to some degree. Especially in extended periodic MMPS systems, where the number of feasible solutions can grow rapidly. Developing new pruning strategies or more aggressive preprocessing rules could shrink the search space further, leading to faster analysis in large-scale or periodic systems.

- **Derive an upper bound on the period length of MMPS systems**

In max-plus and min-plus systems, an upper bound on the possible period length is known. For general MMPS systems, no such bound exists. Establishing one would have multiple benefits: it would limit the search for fixed points and growth rates, improve the efficiency of the periodicity analysis in Chapter 6, and provide theoretical insight into the possible complexity of MMPS dynamics.

- **Perform stability analysis on stochastic MMPS systems**

All results in this thesis are based on deterministic systems, yet many real-world processes have inherent randomness, such as variable travel times or uncertain arrival patterns. Extending the analysis to stochastic MMPS systems would make the theory more applicable to practice. This could involve defining probabilistic stability concepts, adapting MILP formulations to handle uncertainty, or using Monte Carlo methods to assess performance.

- **Integrate control strategies into the transportation framework**

The transportation modelling framework developed here focuses on analysis and simulation but does not yet include active control. Introducing control mechanisms would turn the framework into a decision-support tool. This would also enable direct testing of control algorithms in realistic network scenarios.

- **Add more complex goods-routing logic**

The URS already supports complex routing of goods and materials. Bringing similar functionality into the transportation framework would expand its realism. Making the simulations more reflective of real supply chain and logistics challenges.

- **Integrate the Vehicle Routing Problem (VRP) into the transport framework**

The VRP is a well-known challenge in operations research and is central to many logistics applications. Embedding VRP formulations directly into the transportation framework would allow combined modelling of vehicle scheduling, route selection, and network dynamics. This integration could also support hybrid optimisation approaches that combine MMPS analysis with combinatorial optimisation techniques.

In summary, there are many opportunities to expand both the theoretical scope and the practical usability of the methods developed in this thesis. Some directions aim to deepen the mathematical foundations of MMPS systems, while others seek to push the modelling framework closer to the complexity of real-world operations.

Appendix A

System of Equations for Alternative Transportation Nodes

A-1 End Nodes

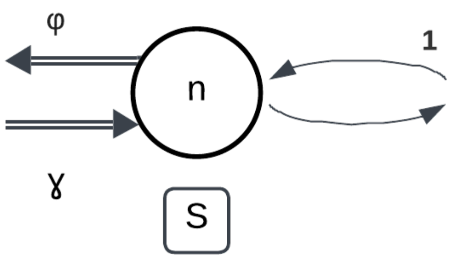


Figure A-1: Visual representation of an in- and output node

Parameter Definitions

Parameter	Definition
$a_{n1}(k)$	Arrival time at node n from node 1
$e_{n1}(k)$	Empty time at node n from node 1
$d_{n1}(k)$	Departure time at node n to node 1
$d_1(k)$	Departure time at node 1 to node n
$s_i(k)$	Stack size of input goods
$s_o(k)$	Stack size of output goods
$\rho(k)$	Load when departing node n to node 1
$\rho_1(k)$	Load when departing node 1 to node n
$\rho_{\max}$	Truck capacity
τ	Travel time from node 1 to node n
u	Unloading speed
L	Loading speed
γ	Inflow rate at node n
φ	Outflow rate at node n

Table A-1: Parameter definition for the state equations of the input and output node

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau \\
e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u} \\
d_{n1}(k) &= \min\left(e_{n1}(k) + \frac{\rho_{\max}}{L}, (L-\gamma)^{-1} \cdot (s_i(k-1) + L \cdot e_{n1}(k) - \gamma \cdot d_{n1}(k-1)\right) \\
s_i(k) &= s_i(k-1) + \gamma \cdot (d_{n1}(k) - d_{n1}(k-1)) - L \cdot (d_{n1}(k) - e_{n1}(k)) \\
s_o(k) &= \max(0, s_o(k-1) + \rho_1(k) - \varphi \cdot (d_{n1}(k) - d_{n1}(k-1)) \\
\rho(k) &= L \cdot (d_{n1}(k) - e_{n1}(k))
\end{aligned}
\tag{A-1}
$$

A-2 Transfer Nodes

General Parameter Definitions for Transfer Nodes and Pass-through Nodes

Parameter	Definition
$a_{n1}(k)$	Arrival time at node n from node 1
$e_{n1}(k)$	Empty time at node n from node 1
$d_{n1}(k)$	Departure time at node n to node 1
$d_1(k)$	Departure time at node 1 to node n
$d_2(k)$	Departure time at node 2 to node n
$a_{n2}(k)$	Arrival time at node n from node 2
$e_{n2}(k)$	Empty time at node n from node 2
$d_{n2}(k)$	Departure time at node n to node 2
$s_{21}(k)$	Stack size of goods to node 1
$s_{12}(k)$	Stack size of goods to node 2
$t_1(k)$	Switching signal: truck 1 takes all goods if it arrives first
$t_2(k)$	Switching signal: truck 2 takes all goods if it arrives first
$\rho_{n1}(k)$	Load when departing node n to node 1
$\rho_{n2}(k)$	Load when departing node n to node 2
$\rho_1(k)$	Load when departing node 1 to node n
$\rho_2(k)$	Load when departing node 2 to node n
τ_1	Travel time from node 1 to node n
τ_2	Travel time from node 2 to node n
u_1	Unloading speed between node n and node 1
u_2	Unloading speed between node n and node 2
L_1	Loading speed between node n and node 1
L_2	Loading speed between node n and node 2
$\rho_{1,\max}$	Capacity on route between node n and node 1
$\rho_{2,\max}$	Capacity on route between node n and node 2
M	A sufficiently large constant

Table A-2: General Parameter Definition for Transfer Nodes and Pass-Through Nodes

Transfer Node with Stack

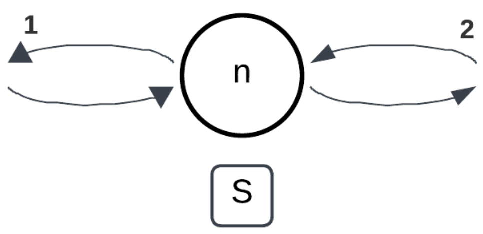


Figure A-2: Visual representation of a transfer node with stack

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau_1 \\
a_{n2}(k) &= d_2(k) + \tau_2 \\
e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \\
e_{n2}(k) &= a_{n2}(k) + \frac{\rho_2(k)}{u_2} \\
s_{21}(k) &= s_{21}(k-1) + \rho_2(k) - L_1\left(d_{n1}(k) - e_{n1}(k)\right) \\
s_{12}(k) &= s_{12}(k-1) + \rho_1(k) - L_2\left(d_{n2}(k) - e_{n2}(k)\right. \\
d_{n1}(k) &= \min(e_{n1}(k) + \frac{\rho_{1,\max}}{L_1}, L_1^{-1}\left(s_{21}(k-1) + \rho_2(k)\right) + e_{n1}(k), \\
&\qquad L_1^{-1}s_{21}(k-1) + e_{n1}(k) + t_1(k)) \\
d_{n2}(k) &= \min\left(e_{n2}(k) + \frac{\rho_{2,\max}}{L_2}, L_2^{-1}\left(s_{12}(k-1) + \rho_1(k)\right) + e_{n2}(k),\right. \\
&\qquad \left. L_2^{-1}s_{12}(k-1) + e_{n2}(k) + t_2(k)\right) \\
t_1(k) &= \max\left(0, (L_1^{-1}S_{21}(k-1) + e_{n1}(k) - a_{n2}(k)) \cdot M\right) \\
t_2(k) &= \max\left(0, \left(L_2^{-1}s_{12}(k-1) + e_{n2}(k) - a_{n1}(k)\right) \cdot M\right) \\
\rho_1(k) &= L_1 \cdot \left(d_{n1}(k) - e_{n1}(k)\right) \\
\rho_2(k) &= L_2 \cdot \left(d_{n2}(k) - e_{n2}(k)\right)
\end{aligned}
\tag{A-2}
$$

Transfer Node with Input

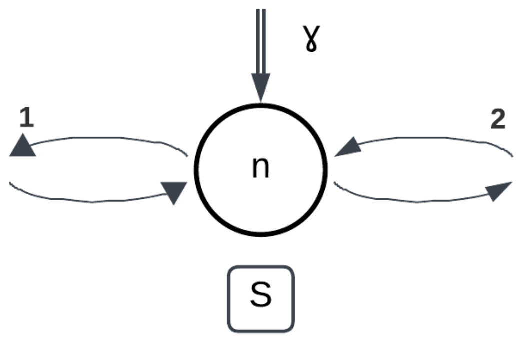


Figure A-3: Visual representation of a transfer node with input

Transfer Node with Input Specific Parameter Definitions

Parameter	Definition
γ	Inflow rate at node n
β	Fraction from input to truck for node 2
$1-\beta$	Fraction from input to truck for node 1

Table A-3: Parameter definition for the state equations of the transfer node with stack and input

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau_1 \\
a_{n2}(k) &= d_2(k) + \tau_2 \\
e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \\
e_{n2}(k) &= a_{n2}(k) + \frac{\rho_2(k)}{u_2} \\
s_{12}(k) &= s_{12}(k-1) + \beta\gamma\left(d_{n2}(k) - d_{n2}(k-1)\right) + \\
&\quad u_1\left(e_{n1}(k) - a_{n1}(k)\right) - L_2\left(d_{n2}(k) - e_{n2}(k)\right) \\
s_{21}(k) &= s_{21}(k-1) + (1-\beta)\gamma\left(d_{n1}(k) - d_{n1}(k-1)\right) + \\
&\quad u_2\left(e_{n2}(k) - a_{n2}(k)\right) - L_1(d_{n1}(k) - e_{n1}(k)) \\
d_{n1}(k) &= \min\Big(e_{n1}(k) + \frac{\rho_{\max}}{L_1}, (L_1 - \beta\gamma)^{-1}(s_{21}(k-1) + L_2 e_{n2}(k) + \\
&\quad u_2\left(e_{n2}(k) - a_{n2}(k)\right) - \beta\gamma d_{n1}(k-1)), \\
&\qquad e_{n1}(k) + \frac{L_1^{-1}}{1 - \frac{\beta\gamma}{L_1}}\left(s_{21}(k-1) + \beta\gamma\left(e_{n1}(k) - d_{n1}(k-1)\right)\right) + t_1(k)\Big) \\
d_{n2}(k) &= \min\Big(e_{n2}(k) + \frac{\rho_{\max}}{L_2}, (L_1 - (1-\beta)\gamma)^{-1}(s_{12}(k-1) + L_1 e_{n1}(k) + \\
&\quad u_1(e_{n1}(k) - a_{n1}(k)) - (1-\beta)\gamma d_{n2}(k-1)), \\
&\qquad e_{n1}(k) + \frac{L_2^{-1}}{1 - \frac{(1-\beta)\gamma}{L_2}}(s_{12}(k-1) + (1-\beta)\gamma(e_{n2}(k) - d_{n2}(k-1)) + t_2(k)\Big) \\
t_1(k) &= \max\Big(0, (e_1(k) - \frac{L_1^{-1}}{1 - \frac{\beta\gamma}{L_1}}\left(s_{21}(k-1) + \beta\gamma\left(e_1(k) - d_1(k-1)\right)\right) - a_2(k))M\Big) \\
t_2(k) &= \max\Big(0, (e_{n1}(k) + \frac{L_2^{-1}}{1 - \frac{(1-\beta)\gamma}{L_2}}(s_{12}(k-1) + (1-\beta)\gamma(e_{n2}(k) - \\
&\quad d_{n2}(k-1))) - a_{n1}(k)) \cdot M\Big)
\end{aligned}
\tag{A-3}
$$

Transfer Node with Output

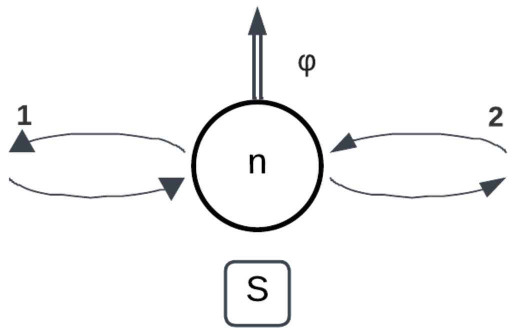


Figure A-4: Visual representation of transfer node with output

Transfer Node with Output Specific Parameter Definitions

Parameter	Definition
φ	Outflow rate of goods at node n
α	Fraction of goods from node 1 to truck 2
$1-\alpha$	Fraction of goods from node 1 to output stack
β	Fraction of goods from node 2 to truck 1
$1-\beta$	Fraction of goods from node 2 to output stack

Table A-4: Parameter definition for the state equations of the transfer node with stack and output

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau_1 \\
a_{n2}(k) &= d_2(k) = \tau_2 \\
e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \\
e_{n2}(k) &= a_{n2}(k) + \frac{\rho_2(k)}{u_2} \\
s_o(k) &= \max\Big(0, s_o(k-1) - \varphi(d_l(k) - d_l(k-1)) + \alpha\rho_1(k) + \beta\rho_2(k)\Big) \\
d_l(k) &= \max(d_1(k), d_2(k)) \\
s_{21}(k) &= s_{21}(k-1) + (1-\beta)\rho_2(k) - L_1(d_{n1}(k) - e_{n1}(k)) \\
s_{12}(k) &= s_{12}(k-1) + (1-\alpha)\rho_1(k) - L_2(d_{n2}(k) - e_{n2}(k)) \\
d_{n1}(k) &= \min\Big(e_{n1}(k) + \frac{\rho_{1,\max}}{L_1}, L_1^{-1}(s_{21}(k-1) + (1-\beta)\rho_2(k)) + e_{n1}, \\
&\qquad L_1^{-1}(s_{21}(k-1)) + e_{n1} + t_1(k)\Big) \\
d_{n2}(k) &= \min\Big(e_{n2}(k) + \frac{\rho_{2,\max}}{L_2}, L_2^{-1}(s_{12}(k-1) + (1-\alpha)\rho_1(k)) + e_{n2}, \\
&\qquad L_2^{-1}(s_{12}(k-1)) + e_{n2} + t_2(k)\Big) \\
t_1(k) &= \max\Big(0, (L_1^{-1}(s_{21}(k-1) + e_{n1}(k) - a_{n2}(k)) \cdot M\Big) \\
t_2(k) &= \max\Big(0, (L_2^{-1}(s_{12}(k-1) + e_{n2}(k) - a_{n1}(k)) \cdot M\Big) \\
\rho_{n1}(k) &= L_1 \cdot (d_{n1}(k) - e_{n1}(k)) \\
\rho_{n2}(k) &= L_2 \cdot (d_{n2}(k) - e_{n2}(k))
\end{aligned}
\tag{A-4}
$$

Transfer Node with Input and Output

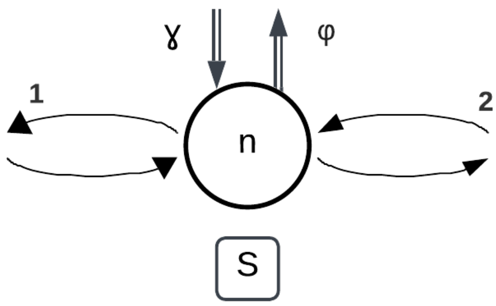


Figure A-5: Visual representation transfer node with input and output

Transfer Node with Input and Output Specific Parameter Definitions

Parameter	Definition
$d_l(k)$	Departure time of last truck
$s_o(k)$	Stack of goods for output
φ	Outflow rate at node n
γ	Inflow rate at node n
α	Fraction from node 1 to truck 2
$1-\alpha$	Fraction from node 1 to output stack
β	Fraction from node 2 to truck 1
$1-\beta$	Fraction from node 2 to output stack
σ	Fraction of input to node 1
$1-\sigma$	Fraction of input to node 2

Table A-5: Parameter definition for the state equations of the transfer node with stack and output

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau_1 \\
a_{n2}(k) &= d_2(k) + \tau_2 \\
e_{n1}(k) &= a_{n1}(k) + \frac{\rho_1(k)}{u_1} \\
e_{n2}(k) &= a_{n2}(k) + \frac{\rho_2(k)}{u_2} \\
\rho_{n1}(k) &= L_1(d_{n1}(k) - e_{n1}(k)) \\
\rho_{n2}(k) &= L_2(d_{n2}(k) - e_{n2}(k)) \\
s_{21}(k) &= s_{21}(k-1) + \sigma\gamma(d_{n1}(k) - d_{n1}(k-1)) + (1-\beta)\rho_2(k) - L_1(d_{n1}(k) - e_{n1}(k)) \\
s_{12}(k) &= s_{12}(k-1) + (1-\sigma)\gamma(d_{n2}(k) - d_{n2}(k-1)) + (1-\alpha)\rho_1(k) - L_2(d_{n2}(k) - e_{n2}(k)) \\
s_o(k) &= \max\Big(0, s_o(k-1) - \varphi(d_l(k) - d_l(k-1)) + \beta\rho_2(k) + \alpha\rho_1(k)\Big) \\
d_l(k) &= \max(d_{n1}(k), d_{n2}(k)) \\
d_{n1}(k) &= \min\Big(a_{n1}(k) + \frac{\rho_{1,\max}}{L_1}, (L_1 - \sigma\gamma)^{-1}(s_{21}(k-1) + (1-\beta)\rho_2(k) - \sigma\gamma d_{n1}(k-1) + \\
&\qquad L_1 e_{n1}(k)), (L_1 - \sigma\gamma)^{-1}(s_{21}(k-1) - \sigma\gamma d_{n1}(k-1) + L_1 e_{n1}(k)) + t_1(k)\Big) \\
d_{n2}(k) &= \min\Big(a_{n2}(k) + \frac{\rho_{2,\max}}{L_2}, (L_2 - \sigma\gamma)^{-1}(s_{12}(k-1) + (1-\alpha)\rho_1(k) - (1-\sigma)\gamma d_{n2}(k-1) \\
&\qquad + L_2 e_{n2}(k)), (L_2 - \sigma\gamma)^{-1}(s_{12}(k-1) - (1-\sigma)\gamma d_{n2}(k-1) + L_2 e_{n2}(k)) + t_2(k)\Big) \\
t_1(k) &= \max\Big(0, (e_{n1}(k) - \frac{L_1^{-1}}{1 - \frac{\beta\gamma}{L_1}}\left(s_{21}\left(k-1\right) + \beta\gamma\left(e_{n1}\left(k\right) - d_{n1}\left(k-1\right)\right)\right) - a_{n2}\left(k\right)\,)M\Big) \\
t_2(k) &= \max\Big(0, (e_{n1}(k) + \frac{L_2^{-1}}{1 - \frac{(1-\beta)\gamma}{L_2}}(s_{12}(k-1) + \\
&\qquad (1-\beta)\gamma(e_{n2}(k) - d_{n2}(k-1))) - a_{n1}(k))M\Big)
\end{aligned}
\tag{A-5}
$$

A-3 Pass-through Nodes

Pass-through Node with Input

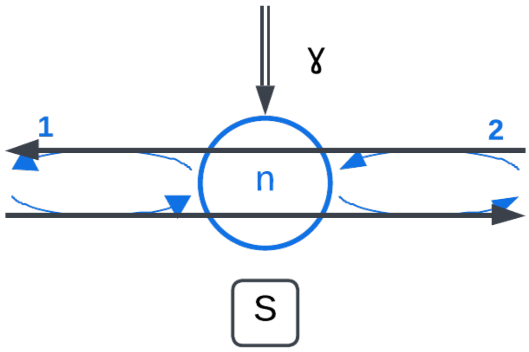


Figure A-6: Visual representation of a pass-through node with input

Pass-through Node with Input Specific Parameter Definitions

Parameter	Definition
γ	Inflow rate at node n
β	Fraction from input to truck 1

Table A-6: Parameter definition for the state equations of the Pass-through node with stack and input

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau_1 \\
a_{n2}(k) &= d_2(t) + \tau_2 \\
\rho_{n1}(k) &= \rho_2(k) + L_2\left(d_{n1}(k) - a_{n2}(k)\right) \\
\rho_{n2}(k) &= \rho_1(k) + L_1\left(d_{n2}(k) - a_{n1}(k)\right) \\
s_{21}(k) &= s_{21}(k-1) + \beta\gamma\left(d_{n1}(k) - d_{n1}(k-1)\right) - L_1\left(d_{n1}(k) - a_{n2}(k)\right) \\
s_{12}(k) &= s_{12}(k-1) + (1-\beta)\gamma\left(d_{n2}(k) - d_{n2}(k-1)\right) - L_2\left(d_{n2}(k) - a_{n1}(k)\right) \\
d_{n1}(k) &= \min\Bigg(a_2(k) + \frac{\rho_{1,\max} - \rho_2(k)}{L_1}, \\
&\qquad (L_1 - \beta\gamma)^{-1}\left(s_{21}(k-1) - \beta\gamma d_{n1}(k-1) + L_1 a_{n2}(k)\right)\Bigg) \\
d_{n2}(k) &= \min\Bigg(a_1(k) + \frac{\rho_{2,\max} - \rho_1(k)}{L_1}, \\
&\qquad (L_2 - (1-\beta)\gamma)^{-1}\left(s_{12}(k-1) - (1-\beta)\gamma d_{n2}(k-1) + L_1 a_{n1}(k)\right)\Bigg)
\end{aligned}
\tag{A-6}
$$

Pass-through Node with Output

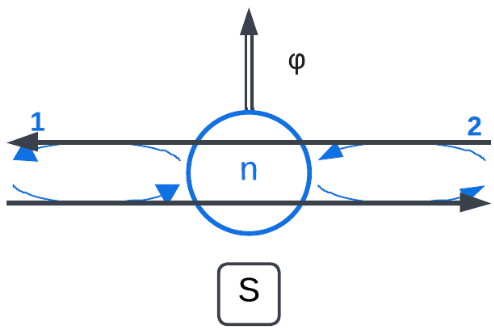


Figure A-7: Visual representation pass-through node with output

Pass-through Node with Output Specific Parameter Definitions

Parameter	Definition
$d_l(k)$	Departure time of last truck
$s_o(k)$	Stack of goods for output
φ	Outflow rate at node n
$1-\alpha$	Fraction of goods from node 2 to truck 1
α	Fraction of goods from node 2 to output stack
$1-\beta$	Fraction of goods from node 1 to truck 2
β	Fraction of goods from node 1 to output stack

Table A-7: Parameter definition for the state equations of the Pass-through node with stack and input

State Equations

$$\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau \\
a_{n2}(k) &= d_2(k) + \tau \\
\rho_{n1}(k) &= (1-\alpha)\rho_2(k) \\
\rho_{n2}(k) &= (1-\beta)\rho_1(k) \\
d_{n1}(k) &= a_{n2}(k) + \frac{\alpha\rho_2(k)}{u_2} \\
d_{n2}(k) &= a_{n1}(k) + \frac{\beta\rho_1(k)}{u_1} \\
d_l(k) &= \max\left(d_{n1}(k), d_{n2}(k)\right) \\
s_o(k) &= \max(0, s_o(k-1) - \varphi\left(d_l(k) - d_l(k-1) + \alpha\rho_2(k) + \beta\rho_1(k)\right)
\end{aligned} \tag{A-7}$$

Pass-through Node with Input and Output

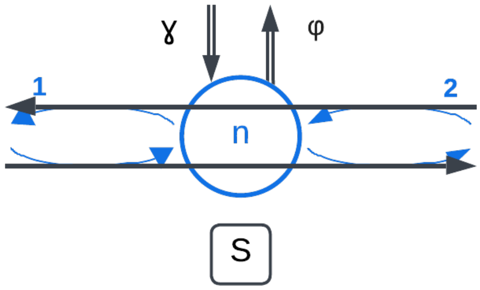


Figure A-8: Visual representation of a pass-through node with input and output

Pass-through Node with Input and Output Specific Parameter Definitions

Parameter	Definition
φ	Outflow rate at node n
γ	Inflow rate at node n
$1-\alpha$	Fraction of goods from node 2 to truck 1
α	Fraction of goods from node 2 to output stack
$1-\beta$	Fraction of goods from node 1 to truck 2
β	Fraction of goods from node 1 to output stack
σ	Fraction of input to node 1
$1-\sigma$	Fraction of input to node 2

Table A-8: Parameter definition for the state equations of the Pass-through node with stack, input and output

State Equations

$$
\begin{aligned}
a_{n1}(k) &= d_1(k) + \tau \\
a_{n2}(k) &= d_2(k) + \tau \\
\rho_{n1}(k) &= \rho_2(k) - u_2\left(e_{n1}(k) - a_{n2}(k)\right) + L_1\left(d_{n1}(k) - e_{n1}(k)\right) \\
\rho_{n2}(k) &= \rho_1(k) - u_1\left(e_{n2}(k) - a_{n1}(k)\right) + L_2\left(d_{n2}(k) - e_{n2}(k)\right) \\
e_{n1}(k) &= a_{n2}(k) + \frac{\alpha\rho_2(k)}{u_2} \\
e_{n2}(k) &= a_{n1}(k) + \frac{\beta\rho_1(k)}{u_1} \\
s_{21}(k) &= s_{21}(k-1) + \sigma\gamma(d_{n1}(k) - d_{n1}(k-1)) - L_1\left(d_{n1}(k) - e_{n2}(k)\right) \\
s_{12}(k) &= s_{12}(k-1) + (1-\sigma)\gamma(d_{n2}(k) - d_{n2}(k-1)) - L_2(d_{n2}(k) - e_{n1}(k)) \\
d_{n1}(k) &= \min\Big(a_{n2}(k) + \frac{\rho_{\max} - (1-\alpha)\rho_2(k)}{L_1}, \\
&\qquad (L_1 - \beta\gamma)^{-1}\left(s_{21}(k-1) - \sigma\gamma d_{n1}(k-1) + L_1 e_{n2}(k)\right)\Big) \\
d_{n2}(k) &= \min\Big(a_{n1}(k) + \frac{\rho_{\max} - (1-\beta)\rho_1(k)}{L_2}, \\
&\qquad (L_2 - (1-\beta)\gamma)^{-1}\left(s_{12}(k-1) - (1-\sigma)\gamma d_{n2}(k-1) + L_1 e_{n1}(k)\right)\Big) \\
d_L(k) &= \max\left(d_{n1}(k), d_{n2}(k)\right) \\
S_o(k) &= \max\Big(0, S_o(k-1) - \varphi(d_l(k) - d_l(k-1) + \alpha\rho_2(k) + \beta\rho_1(k)\Big)
\end{aligned}
\tag{A-8}
$$

Appendix B

System Matrices Example 7.3

$A_{\textsf{full}} =$

$$
\begin{bmatrix}
10 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & 10 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 8 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 8 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon \\
\varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0
\end{bmatrix}
$$

$B_{\textsf{full}} =$

$$\begin{bmatrix}
0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & 1.25 & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0 & \top \\
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & \top & 0
\end{bmatrix}$$

$C_{\textsf{full}} =$

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.028 & -0.142 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

$D_{\textbf{full}} =$

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
40 & 0 & -35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1.142 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-40 & 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.04 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.025 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.025 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.033 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.028 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.028 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.033 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Appendix C

MATLAB: Full MILP Search Algorithm

```
clear
close all


% === Define all the variables and system matrices in ABCD form including
    s ===


% === Create structure matrices for solvability and MILP search route ===

S_A = double((A ~= -inf));
S_B = double((B ~= inf));
S_D = double((D ~= 0));
S_s = S_A * S_B * S_D;


% === locate all branching point for the MILP ===


% Find column positions of 1s in each row
cols_A = arrayfun(@(i) find(S_A(i, :) == 1), (1:size(S_A, 1))', '
    UniformOutput', false);
cols_B = arrayfun(@(i) find(S_B(i, :) == 1), (1:size(S_B, 1))', '
    UniformOutput', false);

% Prepare all data
values   = [sum_A; sum_B];
rows     = [(1:size(S_A,1))'; (1:size(S_B,1))'];
cols     = [cols_A; cols_B];
source   = [repmat("A", size(S_A,1), 1); repmat("B", size(S_B,1), 1)];

```

```
% Create table
search_path = table(values, rows, cols, source, ...
    'VariableNames', {'Value', 'RowIndex', 'ColumnIndices', 'Source'});

% Remove rows where Value == 1 and sort in descending order
search_path = search_path(search_path.Value ~= 1, :);
search_path = sortrows(search_path, 'Value', 'descend');


% === Check solvitility of the system by searching for cycles ===
G = digraph(transpose(S_s));
hasCycle = ~isdag(G);


if hasCycle
    warning('There are cycles in the communications graph.');
else
    disp('There are no cycles in the communications graph.');
end

% === Determine state dependencies for simulation ===

executionOrder = toposort(G);
inDegree = indegree(G);
levels = zeros(size(executionOrder));

% Assign nodes to levels
for i = 1:length(executionOrder)
    node = executionOrder(i);
    predNodes = predecessors(G, node);
    if isempty(predNodes)
        levels(i) = 1;
    else
        levels(i) = max(levels(ismember(executionOrder, predNodes))) + 1;
    end
end

% Group nodes by levels
maxLevel = max(levels);
executionCell = cell(maxLevel, 1);
for lvl = 1:maxLevel
    group = executionOrder(levels == lvl);
%     fprintf('Execution Group %d: %s\n', lvl, num2str(group));
    executionCell{lvl} = group;
end




% === check for time invariance  ===

C_11 = %add submatrices depending on system construction
D_11 = %add submatrices depending on system construction
```

```
C_21 = %add submatrices depending on system construction
D_21 = %add submatrices depending on system construction

CD_11 = [C_11, D_11];
CD_21 = [C_21, D_21];

sum_time_time_invariance = sum(CD_11,2);
sum_quantitity_time_invariance = sum(CD_21,2);

if sum(sum_time_time_invariance) == length(sum_time_time_invariance) && sum(sum_quantitity_time_invariance) == 0
    disp('The system is Time Invariant');
else
    warning('Warning: The system is Not Time Invariant!');
end

% === simulate the system ===
% === Simulation is optional ===

x0 = [];


x_state = zeros(X,k);
x_state(:,1) = x0;


for i = 2:k
    for level = 1:length(executionCell)
        rowsToCompute = executionCell{level};
        x_intermediate = maxplus(A,minplus(B,(C*x_state(:,i-1)+ D*x_state(:,i))));
        x_state(rowsToCompute,i) = x_intermediate(rowsToCompute);
    end
end



% === MILP preprocessing ===

feasible_indices = {};
nonfeasible_indices = {};


for i = 1:height(search_path)
    row_idx = search_path{i, 2};
    col_indices = search_path{i, 3}{:};
    source = search_path{i, 4};

    for j = 1:length(col_indices)
        col_idx = col_indices(j);
        input_cell = {row_idx, col_idx, source};
        fprintf('Trying row %d, col %d, source %s\n', row_idx, col_idx, source);
```

```

        [feasible, lam_opt_single, x_opt_single, ~, ~ , ~, ~] =
           MILP_MMPS_MOSEK_full_search(A,B,C,D,s,input_cell,[],
           nonfeasible_indices);

        if feasible
            fprintf('feasible result at row %d, col %d, source %s\n',
               row_idx, col_idx, source);
            feasible_indices(end+1, :) = {row_idx, col_idx, source};
        else
            fprintf('NON-feasible result at row %d, col %d, source %s\n',
                row_idx, col_idx, source);
            nonfeasible_indices(end+1, :) = {row_idx, col_idx, source};
        end
    end
end

input_table = cell2table(feasible_indices, 'VariableNames', {'RowIndex',
    'ColumnIndices', 'Source'});

% Group by RowIndex and Source
[group_keys, ~, group_ids] = unique(input_table(:, {'RowIndex', 'Source'
    }), 'rows');

% Initialize result containers
Value        = zeros(height(group_keys), 1);
RowIndex     = group_keys.RowIndex;
ColumnIndices  = cell(height(group_keys), 1);
Source       = group_keys.Source;

% For each group, compute value and col block
for i = 1:height(group_keys)
    group_mask = (group_ids == i);
    cols = input_table.ColumnIndices(group_mask);
    ColumnIndices{i} = sort(cols(:)');
    Value(i) = numel(cols);
end

% Build output table
main_table = table(Value, RowIndex, ColumnIndices, Source);

% Split into two tables
split_idx = main_table.Value == 1;
single_value_table = main_table(split_idx, :);
main_table = main_table(~split_idx, :);

% Sort by Value descending
main_table = sortrows(main_table, 'Value', 'descend');
search_path = main_table;

% === run the entire MILP search on search_path  ===
```

```
[feasible_paths, lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,
    p_opt_all, q_opt_all] = search_tree_algorithm_full(single_value_table,
    search_path,A,B,C,D,s);

% === Remove first p and q matrices as they are empty due to matlabs
    tensor filling method ===
p_opt_all = p_opt_all(:,:,2:end);
q_opt_all = q_opt_all(:,:,2:end);
```

```
function [feasible_paths, lambda_opt_all, x_opt_all, y_opt_all, w_opt_all
    , p_opt_all, q_opt_all] = search_tree_algorithm_full(
    single_value_table,search_path,A,B,C,D,s)

% === initialize outputs ===
    feasible_paths = {};
    lambda_opt_all = [];
    x_opt_all = [];
    y_opt_all = [];
    w_opt_all = [];
    p_opt_all = [];
    q_opt_all = [];

    % === Start Depth First Search from level 1 ===
    [feasible_paths,lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,
        p_opt_all ,q_opt_all] = dfs_algorithm(1, [], search_path,
        feasible_paths,lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,
        p_opt_all ,q_opt_all,A,B,C,D,s,single_value_table);

    % === Show all feasible paths found ===
    disp('All feasible paths:');
    for i = 1:length(feasible_paths)
        disp(feasible_paths{i});
    end
end
```

```
function [feasible_paths,lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,
    p_opt_all ,q_opt_all] = dfs_algorithm(level, current_path, search_path
    , feasible_paths,lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,
    p_opt_all ,q_opt_all,A,B,C,D,s,single_value_table)

%=== Add global step counter ===
    persistent dfs_counter
    if isempty(dfs_counter)
        dfs_counter = 0;
    end

    % === terminate route if bottom level of search_path has been reached
        ===
    if level > height(search_path)
        fprintf('Feasible path found:\n');
        disp(current_path);
        feasible_paths{end+1} = current_path;
        return;
```

```
    end

    row_val = search_path.RowIndex(level);
    col_list = search_path.ColumnIndices{level};
    source_val = search_path.Source{level};

    for col_val = col_list
        dfs_counter = dfs_counter + 1;
        fprintf('[Step %d] Trying row %d, col %d, source %s\n', dfs_counter, row_val, col_val, source_val);
        path_try = [current_path; {row_val, col_val, source_val}];

        % === Run MILP ===
         [feasible,lambda_opt, x_opt, y_opt, w_opt, p_opt, q_opt] = MILP_MMPS_MOSEK_full_search(A,B,C,D,s,path_try,single_value_table,[]);
        if level == height(search_path) && feasible
            lambda_opt_all(end+1) = lambda_opt;
            x_opt_all(:,end+1) = x_opt;
            y_opt_all(:,end+1) = y_opt;
            w_opt_all(:,end+1) = w_opt;
            p_opt_all(:,:,end+1) = p_opt;
            q_opt_all(:,:,end+1) = q_opt;
        end

        if feasible
            % === Go deeper and capture updated feasible_paths ===
            [feasible_paths,lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,p_opt_all ,q_opt_all] = dfs_algorithm(level + 1, path_try, search_path, feasible_paths,lambda_opt_all, x_opt_all, y_opt_all, w_opt_all,p_opt_all ,q_opt_all,A,B,C,D,s,single_value_table);
        else
            fprintf('Infeasible at row %d, col %d, source %s - backtracking\n', row_val, col_val, source_val);
        end
    end
end
```

```
function [feasible, lambda_opt, x_opt, y_opt, w_opt, p_opt, q_opt] = MILP_MMPS_MOSEK_full_search(A,B,C,D,s,search_path,single_value_table,non_feasible_input_cells)

    eps = 1e3; % choose own M such that it is large enough

    M_D = max(abs(D(~isinf(D)))) + eps;
    M_C = max(abs(C(~isinf(C)))) + eps;
    M_B = max(abs(B(~isinf(B)))) + eps;
    M_A = max(abs(A(~isinf(A)))) + eps;

    M = max([M_A, M_B, M_C, M_D]);

    [n, m] = size(A);
```

```
    [~, l_dim] = size(B);
    d = D * s;

    % === Variable indexing ===
    num_lambda = 1;
    num_x = n;
    num_y = m;
    num_w = l_dim;
    num_p = m * l_dim;
    num_q = n * m;

    idx_lambda = 1;
    idx_x = @(i) idx_lambda + i;
    idx_y = @(j) idx_lambda + n + j;
    idx_w = @(l) idx_lambda + n + m + l;
    idx_p = @(j,l) idx_lambda + n + m + l_dim + (j-1)*l_dim + l;
    idx_q = @(i,j) idx_lambda + n + m + l_dim + num_p + (i-1)*m + j;

    total_vars = num_lambda + num_x + num_y + num_w + num_p + num_q;

    % === Constraint matrix ===
    Aineq = [];
    bineq = [];
    Aeq = [];
    beq = [];

    % === Constraints 1-2 ===
    for j = 1:m
        for l = 1:l_dim
            if isinf(B(j,l)); continue; end

            % Constraint 1
            row = sparse(1, total_vars);
            row(idx_lambda) = -d(l);
            row(idx_y(j)) = 1;
            row(idx_w(l)) = -1;
            Aineq = [Aineq; row];
            bineq = [bineq; B(j,l)];

            % Constraint 2
            row = sparse(1, total_vars);
            row(idx_lambda) = d(l);
            row(idx_y(j)) = -1;
            row(idx_w(l)) = 1;
            row(idx_p(j,l)) = M;
            Aineq = [Aineq; row];
            bineq = [bineq; -B(j,l) + M];
        end
    end

    % === Constraints 3-4 ===
    for i = 1:n
        for j = 1:m
```

```
            if A(i,j) == -inf; continue; end

            % Constraint 3
            row = sparse(1, total_vars);
            row(idx_lambda) = -s(i);
            row(idx_x(i)) = -1;
            row(idx_y(j)) = 1;
            Aineq = [Aineq; row];
            bineq = [bineq; -A(i,j)];

            % Constraint 4
            row = sparse(1, total_vars);
            row(idx_lambda) = s(i);
            row(idx_x(i)) = 1;
            row(idx_y(j)) = -1;
            row(idx_q(i,j)) = M;
            Aineq = [Aineq; row];
            bineq = [bineq; A(i,j) + M];
        end
    end

    % === Constraint 5: w = (C+D)x ===
    for l = 1:l_dim
        row = sparse(1, total_vars);
        row(idx_w(l)) = 1;
        for i = 1:n
            row(idx_x(i)) = -(C(l,i) + D(l,i));
        end
        Aeq = [Aeq; row];
        beq = [beq; 0];
    end

    % === Constraint 6: sum_j qij = 1 for all i ===
    for i = 1:n
        row = sparse(1, total_vars);
        for j = 1:m
            row(idx_q(i,j)) = 1;
        end
        Aeq = [Aeq; row];
        beq = [beq; 1];
    end


    % === Constraint 7: sum_j pij = 1 for all i ===
    for j = 1:m
        row = sparse(1, total_vars);
        for l = 1:l_dim
            row(idx_p(j,l)) = 1;
        end
        Aeq = [Aeq; row];
        beq = [beq; 1];
    end

```

```
    % === Constraint: fix q and p for the rows known to have one option from the single constraint search ===

    for i = 1:size(single_value_table, 1)
        row_idx = single_value_table{i, 2};
        col_idx = double(single_value_table{i, 3}{1});
        source  = single_value_table{i, 4};

        row = sparse(1, total_vars);  % 1 x total_vars

        if strcmp(source, 'A')
            row(idx_q(row_idx, col_idx)) = 1;
        elseif strcmp(source, 'B')
            row(idx_p(row_idx, col_idx)) = 1;
        else
            error('Unknown source type: %s', source);
        end

        Aeq = [Aeq; row];
        beq = [beq; 1];
    end


    % === Constraint: fix q and p for the rows known to have no option from the single constraint search only in constraints search===

    for i = 1:size(non_feasible_input_cells, 1)
        row_idx = non_feasible_input_cells{i, 1};
        col_idx = double(non_feasible_input_cells{i, 2});
        source  = non_feasible_input_cells{i, 3};

        row = sparse(1, total_vars);  % 1 x total_vars

        if strcmp(source, 'A')
            row(idx_q(row_idx, col_idx)) = 1;
        elseif strcmp(source, 'B')
            row(idx_p(row_idx, col_idx)) = 1;
        else
            error('Unknown source type: %s', source);
        end

        Aeq = [Aeq; row];
        beq = [beq; 0];
    end
    % === Constraint: set q and p rows with search path ===
    for i = 1:size(search_path, 1)
        row_idx = search_path{i, 1};
        col_idx = search_path{i, 2};
        source  = search_path{i, 3};

        row = sparse(1, total_vars);  % 1 x total_vars

        if strcmp(source, 'A')
```

```
            row(idx_q(row_idx, col_idx)) = 1;
        elseif strcmp(source, 'B')
            row(idx_p(row_idx, col_idx)) = 1;
        else
            error('Unknown source type: %s', source);
        end

        Aeq = [Aeq; row];
        beq = [beq; 1];
    end

    % === Constraint 8: Bjl = inf -> pjl = 0 ===
    for j = 1:m
        for l = 1:l_dim
            if isinf(B(j,l))
                row = sparse(1, total_vars);
                row(idx_p(j,l)) = 1;
                Aeq = [Aeq; row];
                beq = [beq; 0];
            end
        end
    end

    % === Constraint 9: Aij = -inf -> qij = 0 ===
    for i = 1:n
        for j = 1:m
            if A(i,j) == -inf
                row = sparse(1, total_vars);
                row(idx_q(i,j)) = 1;
                Aeq = [Aeq; row];
                beq = [beq; 0];
            end
        end
    end



    % === Objective: min lambda ===
    c = zeros(total_vars,1);
    c(idx_lambda) = 1;


    row = sparse(1, total_vars);
            row(idx_lambda) = -1;
            Aineq = [Aineq; row];
            bineq = [bineq; -1];

    % === Integer indices ===
    bin_indices = [];
    for j = 1:m
        for l = 1:l_dim
            bin_indices(end+1) = idx_p(j,l);
        end
```

```
    end
    for i = 1:n
        for j = 1:m
            bin_indices(end+1) = idx_q(i,j);
        end
    end

    % === Variable bounds ===
    blx = -inf(total_vars, 1);
    bux = inf(total_vars, 1);
    blx(bin_indices) = 0;
    bux(bin_indices) = 1;

    % === Build MOSEK model ===
    prob.c = c;
    prob.a = [Aineq; Aeq];
    prob.blc = [-inf(size(bineq)); beq];
    prob.buc = [bineq; beq];
    prob.blx = blx;
    prob.bux = bux;
    prob.ints.sub = bin_indices;

    param.MSK_IPAR_LOG = 1; % 0 to suppress output

    % === Solve ===

    [~, res] = mosekopt('minimize', prob, param);

    if isfield(res, 'sol') && isfield(res.sol, 'int') && isfield(res.sol.int, 'solsta')
        solsta = res.sol.int.solsta;
        prosta = res.sol.int.prosta;

        if strcmp(solsta, 'PRIM_INFEASIBLE') || strcmp(solsta, 'DUAL_INFEASIBLE') || strcmp(solsta, 'UNKNOWN') || ...
            strcmp(prosta, 'PRIM_INFEASIBLE') || strcmp(prosta, 'DUAL_INFEASIBLE') || strcmp(prosta, 'UNKNOWN')
            fprintf('MILP is infeasible or could not be solved. Problem status: %s\n', prosta);
            fprintf('MILP is infeasible or could not be solved. Solution status: %s\n', solsta);
            disp('current search path:')
            disp(search_path)
            lambda_opt = [];
            x_opt = [];
            y_opt = [];
            w_opt = [];
            p_opt = [];
            q_opt = [];
           feasible = false;

        elseif strcmp(solsta, 'INTEGER_OPTIMAL') || strcmp(prosta, 'PRIMAL_FEASIBLE')
```

```
            fprintf('MILP solved optimally.\n');

            x = res.sol.int.xx;

            % === Extract outputs ===
            lambda_opt = x(idx_lambda);
            x_opt = x(idx_x(1:n));
            y_opt = x(idx_y(1:m));
            w_opt = x(idx_w(1:l_dim));

            % Optional outputs
            p_opt = zeros(m, l_dim);
            q_opt = zeros(n, m);
            for j = 1:m
                for l = 1:l_dim
                    p_opt(j,l) = x(idx_p(j,l));
                end
            end

            for i = 1:n
                for j = 1:m
                    q_opt(i,j) = x(idx_q(i,j));
                end
            end
            feasible = true;


        else
            fprintf('Solution status: %s\n', solsta);
            lambda_opt = [];
            x_opt = [];
            y_opt = [];
            w_opt = [];
            p_opt = [];
            q_opt = [];
            feasible = false;
            disp('current search path:')
            disp(search_path)
        end
    else
        fprintf('No solution status returned - likely infeasible or solver failed early.\n');
        error('something went wrong. solver error')
    end
end
```

Appendix D

MATLAB: Generating a System of Equations from an Adjacency Matrix

```
clear
close all

database = database();

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%% input graph and node data %%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


use_example = input('Enter 1 to use the example graph, or 0 to provide your own: ');


if ~use_example
    disp('you have choosen to provide your own system')
    choice = input('Enter 1 to input adjacency matrix manually, 2 to load from a .mat file: ');

    if choice == 1
        graph_input = input('Please enter the adjacency matrix (square matrix): ');
        if ~ismatrix(graph_input) || size(graph_input,1) ~= size(graph_input,2)
            error('Input must be a square matrix.');
        end
        graph = double(graph_input > 0);

    elseif choice == 2
```

```
        [file, path] = uigetfile('*.mat', 'Select the .mat file containing the adjacency matrix');
        if isequal(file,0)
            error('No file selected');
        end
        data = load(fullfile(path, file));

        % Try to find adjacency matrix inside loaded data
        vars = fieldnames(data);
        graph = [];
        for i = 1:length(vars)
            candidate = data.(vars{i});
            if ismatrix(candidate) && size(candidate,1) == size(candidate,2)
                graph = double(candidate > 0);
                fprintf('Using variable "%s" from the file as adjacency matrix.\n', vars{i});
                break;
            end
        end

        if isempty(graph)
            error('No suitable square matrix found in the loaded .mat file.');
        end

    elseif use_example
        error('Invalid choice. Enter 1 or 2.');
    end

else
    disp('you have choosen to use the example graph')
    graph = [0 1 0 1;
             1 0 1 0;
             0 1 0 0;
             1 0 0 0];

    nodes = struct();
    nodes(1).type = 'transfer_without';
    nodes(1).tau1 = 5;
    nodes(1).tau2 = 5;

    nodes(2).type = 'pass_through';
    nodes(2).tau1 = 5;
    nodes(2).tau2 = 5;

    nodes(3).type = 'output';
    nodes(3).outflow = 5;
    nodes(3).tau1 = 5;

    nodes(4).type = 'input';
    nodes(4).inflow = 5;
    nodes(4).tau1 = 5;
```

```
end


sim_input = input('Simulate the system? (true/false): ', 's');
sim_system = strcmpi(sim_input, 'true');
plot_input = input('Plot the system graph? (true/false): ', 's');
plot_system_graph = strcmpi(plot_input, 'true');


if ~exist('nodes', 'var')
    nodes = ensure_nodes_exist(graph);
end

if plot_system_graph
    Network = digraph(graph);

    % Extract node type labels
    labels = arrayfun(@(n) n.type, nodes, 'UniformOutput', false);

    % Plot graph without default labels
    h = plot(Network, ...
        'Layout', 'circle', ...
        'EdgeColor', 'b', ...
        'NodeColor', 'r', ...
        'LineWidth', 1.5, ...
        'MarkerSize', 8, ...
        'NodeLabel', {});

    for i = 1:numel(labels)
        xpos = h.XData(i);
        ypos = h.YData(i);
        text(xpos, ypos, labels{i}, ...
            'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'middle', ...
            'FontSize', 10, ...
            'Rotation', 0, ...
            'Interpreter', 'none');
    end
    title('Graph of system network');
end

%check if graph can be used with current blocks
A_upper = triu(graph, 1);
deg = sum(A_upper, 2);
nodes_more_than_3 = find(deg > 3);

if ~isempty(nodes_more_than_3)
    disp('Nodes connected to more than 3 others (in one direction only):'
        );
    disp(nodes_more_than_3);
    error('there are nodes with connected to more than 3 other nodes.
        There is no building block for this in the toolbox')

```

```
end

%check if all node data is present and nodes are connected propperly

validate_node_info(nodes, graph)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%% ask user for truck data %%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N = size(graph, 1);
[arc_i, arc_j] = find(A_upper);  % List of unique undirected arcs
% Initialize connection lists
for n = 1:N
    nodes(n).connections = {};
end

% Build the connection lists for each node
%first come first serve.
for idx = 1:length(arc_i)
    i = arc_i(idx);
    j = arc_j(idx);

    % Add j to i's connections and vice versa
    nodes(i).connections{end+1} = j;
    nodes(j).connections{end+1} = i;
end

num_arcs = length(arc_i);
truck_id = 1;
arc_to_truck = containers.Map();
truck_arcs = {};

used_arcs = false(num_arcs, 1);

for k = 1:num_arcs
    if used_arcs(k), continue; end

    i = arc_i(k);
    j = arc_j(k);

    key1 = sprintf('%d_%d', i, j);
    key2 = sprintf('%d_%d', j, i);

    % Check if one node is a pass-through
    if isfield(nodes(j), 'type') && startsWith(nodes(j).type, 'pass-
        through')
        % Find other neighbors of j
        neighbors = find(graph(j, :) == 1);
        neighbors(neighbors == i) = [];  % exclude i
```

```matlab

        if length(neighbors) ~= 1
            error('Pass-through node must have exactly two neighbors');
        end

        k2 = neighbors(1);  % the other arc
        % Create arc list for this truck
        key3 = sprintf('%d_%d', j, k2);

        % Find index in arc list
        idx2 = find((arc_i == min(j,k2)) & (arc_j == max(j,k2)));

        if isempty(idx2)
            error('Expected arc not found in arc list');
        end

        used_arcs([k, idx2]) = true;

        arc_to_truck(key1) = truck_id;
        arc_to_truck(key2) = truck_id;
        arc_to_truck(key3) = truck_id;
        arc_to_truck(sprintf('%d_%d', k2, j)) = truck_id;

        truck_arcs{truck_id} = [i j; j k2];  % store arcs

    else
        % Regular arc
        used_arcs(k) = true;
        arc_to_truck(key1) = truck_id;
        arc_to_truck(key2) = truck_id;
        truck_arcs{truck_id} = [i j];
    end

    truck_id = truck_id + 1;
end

% get truck parameters from user
truck_params = struct();
for t = 1:length(truck_arcs)
    fprintf('\nTruck %d connects the following arcs:\n', t);
    arcs = truck_arcs{t};
    for a = 1:size(arcs,1)
        fprintf('  Node %d <--> Node %d\n', arcs(a,1), arcs(a,2));
    end

    cap = input('Enter capacity of truck: ');
    load = input('Enter loading speed: ');
    unload = input('Enter unloading speed: ');

    truck_params(t).cap = cap;
    truck_params(t).load = load;
    truck_params(t).unload = unload;
end
```

```matlab

%%%% Store truck info into node structs


for i = 1:N
    % Find arcs connected to this node
    neighbors = find(graph(i,:) == 1);
    trucks = [];

    for j = neighbors
        key = sprintf('%d_%d', i, j);
        if isKey(arc_to_truck, key)
            trucks(end+1) = arc_to_truck(key);
        end
    end

    nodes(i).truck_ids = unique(trucks);
    nodes(i).num_trucks = length(nodes(i).truck_ids);

    % Optionally store full parameters
    for k = 1:nodes(i).num_trucks
        tid = nodes(i).truck_ids(k);
        nodes(i).truck(k) = truck_params(tid);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%    construct A B C D matrices    %%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


[A_bigg, B_bigg, C_bigg, D_bigg,state_order] = build_transport_network(
    nodes, database);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%  Simulate system  %%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


x0 = zeros(size(A_bigg,1),1);

num_steps = 10;

if sim_system
    state = simulate_system(A_bigg,B_bigg,C_bigg,D_bigg,x0,num_steps);
    plot_state_evolution(state, state_order);
end

```

```
function state = simulate_system(A,B,C,D,x0,num_steps)

    k = num_steps;

    S_A = double((A ~= -inf));
    S_B = double((B ~= inf));
    S_D = double((D ~= 0));
    S_s = S_A * S_B * S_D;

    G = digraph(transpose(S_s));
    hasCycle = ~isdag(G);

    %check if the system is solvable
    if hasCycle
        error('There are cycles in the communications graph.');
    else
        disp('There are no cycles in the communications graph.');
    end

    executionOrder = toposort(G);
    levels = zeros(size(executionOrder));

    for i = 1:length(executionOrder)
        node = executionOrder(i);
        predNodes = predecessors(G, node); % Get parent nodes
        if isempty(predNodes)
            levels(i) = 1;
        else
            levels(i) = max(levels(ismember(executionOrder, predNodes))) + 1;
        end
    end

    maxLevel = max(levels);
    executionCell = cell(maxLevel, 1);
    for lvl = 1:maxLevel
        group = executionOrder(levels == lvl);
        executionCell{lvl} = group;
    end


    state = zeros(size(A,1),k);
    state(:,1) = x0;
    for i = 2:k
        for level = 1:length(executionCell)
            rowsToCompute = executionCell{level};
            x_intermediate = maxplus(A,minplus(B,(C*state(:,i-1)+ D*state(:,i))));
            state(rowsToCompute,i) = x_intermediate(rowsToCompute);
        end
    end
```

```matlab
end

function required_fields = required_fields_list()
    required_fields = containers.Map();
    required_fields('central') = {'inflow','tau1','tau2','tau3'};
    required_fields('input') = {'inflow','tau1'};
    required_fields('output') = {'outflow','tau1'};
    required_fields('in and output') = {'inflow', 'outflow','tau1'};

    required_fields('transfer_with') = {'tau1','tau2'};
    required_fields('transfer_without') = {'tau1','tau2'};
    required_fields('transfer_input') = {'inflow', 'beta','tau1','tau2'};
    required_fields('transfer_output') = {'outflow', 'alpha', 'beta','tau1','tau2'};
    required_fields('transfer_in_and_output') = {'inflow', 'outflow', 'alpha', 'beta', 'sigma','tau1','tau2'};

    required_fields('pass_through') = {'tau1','tau2'};
    required_fields('pass_through_input') = {'inflow', 'beta','tau1','tau2'};
    required_fields('pass_through_output') = {'outflow', 'alpha', 'beta', 'tau1','tau2'};
    required_fields('pass_through_in_and_output') = {'inflow', 'outflow', 'alpha', 'beta', 'sigma','tau1','tau2'};
end

function validate_node_info(nodes, graph)
    required_fields = required_fields_list();
    degrees = sum(graph, 2);

    for i = 1:numel(nodes)
        node = nodes(i);

        % Type check
        if ~isfield(node, 'type') || isempty(node.type)
            error('Node %d is missing the "type" field or it is empty.', i);
        end

        % Normalize type
        node_type = lower(strtrim(node.type));

        % Validate type
        if ~isKey(required_fields, node_type)
            valid_types = strjoin(keys(required_fields), ', ');
            error('Node %d has unrecognized type "%s". Valid types: %s', ...
                  i, node.type, valid_types);
        end

        % Check required fields
        expected = required_fields(node_type);
        for j = 1:numel(expected)
```

```
            fname = expected{j};
            if ~isfield(node, fname) || isempty(node.(fname))
                error('Node %d (type "%s") is missing or has empty required field "%s".', ...
                    i, node.type, fname);
            end
        end

        % Connectivity rule check
        deg = degrees(i);
        if startsWith(node_type, 'pass_through') || startsWith(node_type, 'transfer')
            if deg ~= 2
                error('Node %d (type "%s") must be connected to exactly 2 nodes, but is connected to %d.', ...
                    i, node_type, deg);
            end
        elseif strcmp(node_type, 'central')
            if deg ~= 3
                error('Node %d (type "central") must be connected to exactly 3 nodes, but is connected to %d.', ...
                    i, deg);
            end
        elseif ismember(node_type, {'input', 'output', 'in and output'})
            if deg ~= 1
                error('Node %d (type "%s") must be connected to exactly 1 node, but is connected to %d.', ...
                    i, node_type, deg);
            end
        end
    end
    disp('All nodes passed validation.');
end


function M_filled = fill_template(M_template, params)
    % Replaces symbolic fields in the matrix template using the node parameters
    if isa(M_template, 'sym')
        syms_list = symvar(M_template);

        for i = 1:length(syms_list)
            sym_name = char(syms_list(i));
            replaced = false;

            % ===  Node-level parameter ===
            if isfield(params, sym_name)
                val = params.(sym_name);
                M_template = subs(M_template, syms_list(i), val);
                replaced = true;

            % === Truck parameters (Lk, uk, rho_mk) ===
            else
```

```
                % Match Lk (loading), uk (unloading), rho_mk (capacity)
                tokens = regexp(sym_name, '^(L|u|rho_m)(\d+)$', 'tokens');
                if ~isempty(tokens)
                    prefix = tokens{1}{1};  % L, u, or rho_m
                    idx = str2double(tokens{1}{2});  % truck index
                    if isfield(params, 'truck') && length(params.truck) >= idx
                        truck = params.truck(idx);
                        switch prefix
                            case 'L'
                                val = truck.load;
                            case 'u'
                                val = truck.unload;
                            case 'rho_m'
                                val = truck.cap;
                            otherwise
                                error(['Unknown symbolic truck field: ' sym_name]);
                        end
                        M_template = subs(M_template, syms_list(i), val);
                        replaced = true;
                    else
                        error(['Truck index ' num2str(idx) ' not found in params.truck']);
                    end
                end
            end

            if ~replaced
                error(['Missing parameter for symbol: ' sym_name]);
            end
        end

        M_filled = double(M_template);
    else
        M_filled = M_template;
    end
end


function [A_bigg, B_bigg, C_bigg, D_bigg, state_order_all] = build_transport_network(nodes, database)
    E = -inf;
    T = inf;
    state_order_all = {};

    N = length(nodes);
    A_cells = {};
    B_cells = {};
    C_cells = {};
    D_cells = {};
    F_cells = {};
```

```
    H_cells = {};
    K_cells = {};
    L_cells = {};
    filled_Es = {};
    arm_counts = zeros(N, 1);

    for i = 1:N
        node = nodes(i);
        type = node.type;
        template = database.(type);

        A = fill_template(template.A, node);
        B = fill_template(template.B, node);
        C = fill_template(template.C, node);
        D = fill_template(template.D, node);


        A_cells{end+1} = A;
        B_cells{end+1} = B;
        C_cells{end+1} = C;
        D_cells{end+1} = D;
        F_cells{end+1} = template.F;
        H_cells{end+1} = template.H;


        ordered_states = template.state_order;
        for k = 1:length(ordered_states)
            ordered_states{k} = [ordered_states{k}, '_', num2str(i)];
        end
        state_order_all = [state_order_all; ordered_states(:)];

        % Count arms and store filled E matrices
        k = 1;
        while isfield(template, ['E', num2str(k)])
            Ek = fill_template(template.(['E', num2str(k)]), node);
            filled_Es{i, k} = Ek;
            k = k + 1;
        end
        arm_counts(i) = k - 1;
    end

    % Assemble block diagonal matrices
    A_big = assemble_block_diag(A_cells, E);
    B_big = assemble_block_diag(B_cells, T);
    C_big = blkdiag(C_cells{:});
    D_big = blkdiag(D_cells{:});
    F_big = assemble_block_diag(F_cells, E);
    H_big = assemble_block_diag(H_cells, T);


%%%%%%%%%%%%%%%% construction of E %%%%%%%%%%%%%%%%

    num_nodes = length(nodes);
```

```
    total_cols = 2 * sum(arm_counts);  % 2 columns per arm
    total_rows = size(D_big,1);

    E_big = zeros(total_rows, total_cols);

    % Compute row offset for each node's block in E_big
    row_offsets = cumsum([0; cellfun(@(c) size(c,1), C_cells)']);

    % Assign global arm indices
    global_arm_counter = 0;
    global_col_indices = cell(N, 1);
    for i = 1:N
        global_col_indices{i} = global_arm_counter + (1:arm_counts(i));
        global_arm_counter = global_arm_counter + arm_counts(i);
    end

    % Place E blocks
    for i = 1:num_nodes
        row_start = row_offsets(i) + 1;
        row_end = row_offsets(i+1);

        for a = 1:arm_counts(i)
            conn_node = nodes(i).connections{a};
            conn_arms = nodes(conn_node).connections;
            conn_arm = find(cellfun(@(x) isequal(x, i), conn_arms));


            if isempty(conn_arm)
                error("Could not find reciprocal connection from node %d
                   to node %d", conn_node, i);
            end

            % global column index of that arm
            col_index = global_col_indices{conn_node}(conn_arm);
            col_start = (col_index - 1) * 2 + 1;
            col_end = col_index * 2;

            % insert E block
            E_block = filled_Es{i, a};
            E_big(row_start:row_end, col_start:col_end) = E_block;
        end
    end


%%%%%%%%%%%%%%%% construct K and L %%%%%%%%%%%%%%%%%

    end_nodes_passthrough = find_pass_through_endpoints_with_arms(nodes);


    for i = 1:N
        num_arms = length(nodes(i).connections);
        conn_nodes = cell2mat(nodes(i).connections);
        node = nodes(i);
```

```
        type = node.type;
        KLs = database.(type).KL;
        connected_nodes_types = {nodes(conn_nodes).type};

        node_K_blocks = {};
        node_L_blocks = {};
      connects_to_pass_through = any(cellfun(@(s)  startsWith(s, 'pass_through'), connected_nodes_types));

        for a = 1:num_arms
            target_node = conn_nodes(a);
            KL_block = KLs{a};
            zero_block = zeros(size(KL_block));

            if startsWith(type, 'pass_through')
                node_K_blocks{end+1,1} = zero_block;
                node_L_blocks{end+1,1} = KL_block;

            elseif connects_to_pass_through
                % Check if (node i, arm a) appears in first two columns of any row in end_nodes_passthrough
                is_K_side = any(all(end_nodes_passthrough(:,1:2) == [i, a], 2));

                if is_K_side
                    node_K_blocks{end+1,1} = KL_block;
                    node_L_blocks{end+1,1} = zero_block;
                else
                    node_K_blocks{end+1,1} = zero_block;
                    node_L_blocks{end+1,1} = KL_block;
                end

            else
                % Regular node
                if target_node < i
                    node_K_blocks{end+1,1} = KL_block;
                    node_L_blocks{end+1,1} = zero_block;
                else
                    node_K_blocks{end+1,1} = zero_block;
                    node_L_blocks{end+1,1} = KL_block;
                end
            end
        end

        % Stack vertically for this node
        K_block = vertcat(node_K_blocks{:});
        L_block = vertcat(node_L_blocks{:});

        % Store block-diagonally
        K_cells{end+1} = K_block;
        L_cells{end+1} = L_block;
    end

```

```
    % Build full block-diagonal matrices
    K_big = blkdiag(K_cells{:});
    L_big = blkdiag(L_cells{:});



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% concatinate all matrices %%%%%%%%%%%%%%

    [mA, nA] = size(A_big);
    [mF, nF] = size(F_big);


    A_bigg = -Inf(mA + mF, nA + nF);
    A_bigg(1:mA, 1:nA) = A_big;
    A_bigg(mA+1:end, nA+1:end) = F_big;



    [mB, nB] = size(B_big);
    [mH, nH] = size(H_big);
    B_bigg = Inf(mB + mH, nB + nH);
    B_bigg(1:mB, 1:nB) = B_big;
    B_bigg(mB+1:end, nB+1:end) = H_big;

    [~, nE] = size(E_big);
    [mL, ~] = size(L_big);
    Z = zeros(mL, nE);

    C_bigg = [C_big, zeros(size(E_big)); K_big,Z];
    D_bigg = [D_big, E_big; L_big, Z];

end

function M_big = assemble_block_diag(blocks, default_val)

    total_rows = 0;
    total_cols = 0;
    n_blocks = numel(blocks);
    rows = zeros(n_blocks, 1);
    cols = zeros(n_blocks, 1);

    for i = 1:n_blocks
        [rows(i), cols(i)] = size(blocks{i});
        total_rows = total_rows + rows(i);
        total_cols = total_cols + cols(i);
    end

    M_big = default_val * ones(total_rows, total_cols);

    row_offset = 0;
    col_offset = 0;
    for i = 1:n_blocks
```

```
        r = rows(i);
        c = cols(i);
        M_big(row_offset + (1:r), col_offset + (1:c)) = blocks{i};
        row_offset = row_offset + r;
        col_offset = col_offset + c;
    end
end

function end_node_info = find_pass_through_endpoints_with_arms(nodes)
    is_pass = arrayfun(@(n) startsWith(n.type, 'pass_through'), nodes);
    visited = false(1, length(nodes));
    end_node_info = zeros(0, 4);  % [node1, arm1, node2, arm2]

    for i = 1:length(nodes)
        if visited(i) || is_pass(i)
            continue;
        end

        for a = 1:length(nodes(i).connections)
            path = i;
            current = nodes(i).connections{a};

            if is_pass(current)
                % Follow pass-through chain
                prev = i;
                while is_pass(current)
                    path(end+1) = current;
                    visited(current) = true;
                    next_candidates = setdiff([nodes(current).connections
                        {:}], prev);
                    if isempty(next_candidates)
                        break;
                    end
                    prev = current;
                    current = next_candidates(1);  % follow one direction
                end
                path(end+1) = current;

                % Determine arms
                node1 = path(1);
                node2 = path(end);

                arm1 = find(cell2mat(nodes(node1).connections) == path(2)
                    );
                arm2 = find(cell2mat(nodes(node2).connections) == path(
                    end-1));

                if isempty(arm1) || isempty(arm2)
                    warning("Could not identify arms for nodes %d and %d
                        ", node1, node2);
                    continue;
                end

```

```
                pair = sort([node1, node2]);
                % Ensure unique entry (order-independent)
                existing = (end_node_info(:,1) == pair(1) & end_node_info(:,3) == pair(2)) | ...
                           (end_node_info(:,1) == pair(2) & end_node_info(:,3) == pair(1));
                if ~any(existing)
                    if node1 <= node2
                        end_node_info(end+1, :) = [node1, arm1, node2, arm2];
                    else
                        end_node_info(end+1, :) = [node2, arm2, node1, arm1];
                    end
                end
            end
        end
    end
end


function plot_state_evolution(state, state_order)
    num_tracked_states = numel(state_order);
    state = state(1:num_tracked_states, :);  % Trim off extra non-relevant rows

    % Prepare groups
    groups = struct( ...
        'arrivals', struct('indices', [], 'label', 'Arrivals', 'prefix', 'a'), ...
        'departures', struct('indices', [], 'label', 'Departures', 'prefix', 'd'), ...
        'loads', struct('indices', [], 'label', 'Loads', 'prefix', 'rho'), ...
        'stacks', struct('indices', [], 'label', 'Stacks', 'prefix', 's') ...
    );

    % Assign each state to a group
    for i = 1:num_tracked_states
        name = state_order{i};
        if startsWith(name, 'a')
            groups.arrivals.indices(end+1) = i;
        elseif startsWith(name, 'd')
            groups.departures.indices(end+1) = i;
        elseif startsWith(name, 'rho')
            groups.loads.indices(end+1) = i;
        elseif startsWith(name, 's')
            groups.stacks.indices(end+1) = i;
        end
    end

    % Helper function to plot a group
```

```
    function plot_group(group)
        if isempty(group.indices), return; end
        figure('Name', group.label);
        hold on;
        for idx = group.indices
            plot(state(idx, :), 'LineWidth', 1, 'DisplayName',
                state_order{idx});
        end
        xlabel('Cycle');
        ylabel('Time');
        title([group.label ' over Time']);
        legend('Location', 'best');
        grid on;
        hold off;
    end

    % Plot all groups
    plot_group(groups.arrivals);
    plot_group(groups.departures);
    plot_group(groups.loads);
    plot_group(groups.stacks);
end

function nodes = ensure_nodes_exist(graph)
    % Only run if nodes is empty or missing fields
    fprintf('No node data found. Starting interactive setup.\n');
    nodes = struct();
    required_fields = required_fields_list();
    degrees = sum(graph, 2);
    num_nodes = length(degrees);


    for i = 1:num_nodes
        deg = degrees(i);
        fprintf('\nConfiguring Node %d (degree %d)\n', i, deg);

        % Suggest valid node types based on degree
        if deg == 1
            suggestions = {'input', 'output', 'in and output'};
        elseif deg == 2
            suggestions = {'transfer_without', 'transfer_with', '
                pass_through', ...
                              'transfer_input', 'transfer_output', '
                                  transfer_in_and_output', ...
                              'pass_through_input', 'pass_through_output', '
                                  pass_through_in_and_output'};
        elseif deg == 3
            suggestions = {'central'};
        else
            error('Node %d has invalid degree %d (only 1, 2 or 3
                supported).', i, deg);
        end

```

```
        fprintf('Possible types based on degree %d: %s\n', deg, strjoin(
            suggestions, ', '));
        % Use menu to avoid typing errors
        indx = menu(sprintf('Select type for node %d (degree = %d):', i,
            deg), suggestions{:});
        if indx == 0
            error('User cancelled node type selection.');
        end
        node_type = suggestions{indx};
        nodes(i).type = node_type;
        fields = required_fields(node_type);

        for j = 1:numel(fields)
            field_name = fields{j};
            val = input(sprintf('Enter value for %s: ', field_name));
            nodes(i).(field_name) = val;
        end
    end
end
```

```
function database = database()

syms tau1 tau2 tau3 rho_m1 L1 L2 L3 u1 u2 u3 outflow inflow beta1 beta2
    beta3
aplha  = sym('aplha');
beta = sym('beta');
sigma = sym('sigma');

T = inf;
E = -inf;


input.A = [ tau1, E,E,E;
            E, 0,E,E;
            E,E,0,E;
            E,E,E,0];

input.B = [ 0,T,T,T,T;
            T,0,T,T,T;
            T,T,rho_m1/L1,0,T;
            T,T,T,T,0];

input.C = [ 0,0,0,0;
            0,1,-inflow,0;
            0,0,0,0;
            0,1/(L1-inflow),-inflow/(L1-inflow),0;
            0,0,0,0];

input.D = [ 0,0,0,0;
            L1,0,inflow-L1,0;
            1,0,0,0;
            L1/(L1-inflow),0,0,0;
            -L1,0,L1,0];
```

```
input.E1 = [     1,0;
                 0,0;
                 0,0;
                 0,0;
                 0,0];
input.state_order = {'a1';'s1';'d1';'rho1'};
input.F = [0,E;E,0];
input.H = [0,T;T,0];
input.KL{1} = [0,0,1,0;0,0,0,1];


output.A = [tau1, E,E,E,E;
            E,0,0,E,E;
            E,E,E,0,E;
            E,E,E,E,0];
output.B = [0,T,T,T,T;
            T,0,T,T,T;
            T,T,0,T,T;
            T,T,T,0,T;
            T,T,T,T,0];
output.C = [0,0,0,0;
            0,1,outflow,0;
            0,0,0,0;
            0,0,0,0;
            0,0,0,0];
output.D = [0,0,0,0;
            0,0,-outflow,0;
            0,0,0,0;
            1,0,0,0;
            0,0,0,0];
output.E1 = [1,0;
            0,1;
            0,0;
            0,1/u1;
            0,0];
output.state_order = {'a1';'s1';'d1';'rho1'};
output.F = [0,E;E,0];
output.H = [0,T;T,0];
output.KL{1} = [0,0,1,0;0,0,0,1];


transfer_without.A = [tau1,E,E,E,E,E,E,E,E,E;
                      E,0,E,E,E,E,E,E,E,E;
                      E,E,0,0,E,E,E,E,E,E;
                      E,E,E,E,0,E,E,E,E,E
                      E,E,E,E,E,tau2,E,E,E,E;
                      E,E,E,E,E,E,0,E,E,E;
                      E,E,E,E,E,E,E,0,0,E;
                      E,E,E,E,E,E,E,E,E,0
                      ];

transfer_without.B = [   0,T,T,T,T,T,T,T,T,T;
                         T,0,T,T,T,T,T,T,T,T;
```

```
                            T,T,0,T,T,T,T,T,T,T;
                            T,T,T,0,T,T,T,T,T,T;
                            T,T,T,T,0,T,T,T,T,T;
                            T,T,T,T,T,0,T,T,T,T;
                            T,T,T,T,T,T,0,T,T,T;
                            T,T,T,T,T,T,T,0,T,T
                            T,T,T,T,T,T,T,T,0,T
                            T,T,T,T,T,T,T,T,T,0];
transfer_without.C = zeros(10,8);
transfer_without.D = [   0,0,0,0,0,0,0,0;
                            1,0,0,0,0,0,0,0;
                            0,1,0,0,0,0,0,0;
                            0,0,0,0,0,1,0,0;
                            0,0,0,0,0,0,0,0;
                            0,0,0,0,0,0,0,0;
                            0,0,0,0,1,0,0,0;
                            0,1,0,0,0,0,0,0;
                            0,0,0,0,0,1,0,0;
                            0,0,0,0,0,0,0,0];

transfer_without.E1 = [  1,0;
                            0,1/u1;
                            0,0;
                            0,0;
                            0,0;
                            0,0;
                            0,0;
                            0,1/L2;
                            0,1/L2;
                            0,1];
transfer_without.E2 = [  0,0;
                            0,0;
                            0,1/L1;
                            0,1/L1;
                            0,1;
                            1,0;
                            0,1/u2;
                            0,0;
                            0,0;
                            0,0];

transfer_without.state_order = {'a1';'e1';'d1';'rho1';'a2';'e2';'d2';'
    rho2'};
transfer_without.F = [0,E,E,E;E,0,E,E;E,E,0,E;E,E,E,0];
transfer_without.H = [0,T,T,T;T,0,T,T;T,T,0,T;T,T,T,0];
transfer_without.KL{1} = [  0,0,1,0,0,0,0,0;
                            0,0,0,1,0,0,0,0;];

transfer_without.KL{2} = [0,0,0,0,0,0,1,0;
                            0,0,0,0,0,0,0,1];
```

```
center_without.A = [tau1,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,tau2,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,tau3,E,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,E,E,E,E,E,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,0,0,0,0,E,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,E,E,E,E,E,E;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,E,E,0,0,0;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,E,0,0,0;
                    E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,0,0,0,0;];


center_without.B = [0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
```

```
                    T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,0;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,0,T;
                    T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,0,T,0];
center_without.C = zeros(27,15);
center_without.D = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                    0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
                    0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
                    0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
                    0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
                    0,0,0,0,0,0,0,0,0,0,0,0,1,0,0;
                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                    0,0,0,0,1,0,0,0,0,0,0,0,0,0,0;
                    0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
                    0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
```

```
                        0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,0,0,0,0,1,0;
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,1,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
                        0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
                        0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,1;
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                        0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
                        0,0,0,0,0,0,0,0,0,1,0,0,0,0,0];

center_without.E1 = [1,0;
                        0,1/u1;
                        0,0;
                        0,0;
                        0,0;
                        0,0;
                        0,0;
                        0,0;
                        0,0;
                        0,0;
                        0,beta1/L2;
                        0,beta1/L2;
                        0,0;
                        0,0;
                        0,beta1/L2;
                        0,beta1;
                        0,0;
                        0,0;
                        0,(1-beta1)/L3;
                        0,(1-beta1)/L3;
                        0,0;
                        0,0;
                        0,(1-beta1)/L3;
                        0,1-beta1;
                        0,0;
                        0,0;
                        0,0];

center_without.E2 = [0,0;0,0;0,beta2/L1;0,beta2/L1;0,0;0,0;0,beta2/L1;0,
    beta2;
     1,0;0,1/u2;0,0;0,0;0,0;0,0;0,0;0,0;0,0;0,0;0,0;
0,0;0,(1-beta2)/L3;0,(1-beta2)/L3;0,(1-beta2)/L3;0,1-beta2;0,0;0,0;0,0];


center_without.E3 = [0,0;
                        0,0;
                        0,0;
```

```
                      0,0;
                      0,beta3/L1;
                      0,beta3/L1;
                      0,beta3/L1;
                      0,beta3;
                      0,0;
                      0,0;
                      0,0;
                      0,0;
                      0,(1-beta3)/L2;
                      0,(1-beta3)/L2;
                      0,(1-beta3)/L2;
                      0,(1-beta3);
                      1,0;
                      0,1/u3;
                      0,0;
                      0,0;
                      0,0;
                      0,0;
                      0,0;
                      0,0;
                      0,0;
                      0,0;
                      0,0];

center_without.state_order = {'a1';'e1';'d1';'rho1';'a2';'e2';'d2';'rho2'
    ;'a3';'e3';'d3';'rho3';'delta1';'delta2';'delta3'};
center_without.F = [0,E,E,E,E,E;E,0,E,E,E,E;E,E,0,E,E,E;E,E,E,0,E,E;E,E,E
    ,E,0,E;E,E,E,E,E,0];
center_without.H = [0,T,T,T,T,T;T,0,T,T,T,T;T,T,0,T,T,T;T,T,T,0,T,T;T,T,T
    ,T,0,T;T,T,T,T,T,0];
center_without.KL{1} = [    0,0,1,0,0,0,0,0,0,0,0,0,0,0,0;
                         0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;];

center_without.KL{2} = [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0;
                         0,0,0,0,0,0,0,1,0,0,0,0,0,0,0;];

center_without.KL{3} = [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0;
                         0,0,0,0,0,0,0,0,0,0,0,1,0,0,0];



 pass_through.A = [ tau1,E,E,E,E,E;
                    E,0,E,E,E,E;
                    E,E,0,E,E,E;
                    E,E,E,tau2,E,E;
                    E,E,E,E,0,E;
                    E,E,E,E,E,0];

 pass_through.B = [  0,T,T,T,T,T;
                    T,0,T,T,T,T;
                    T,T,0,T,T,T;
                    T,T,T,0,T,T;
```

```
                    T,T,T,T,0,T;
                    T,T,T,T,T,0];
pass_through.C = zeros(6,6);

pass_through.D = [   0,0,0,0,0,0;
                     0,0,0,1,0,0;
                     0,0,0,0,0,0;
                     0,0,0,0,0,0;
                     1,0,0,0,0,0;
                     0,0,0,0,0,0;];

pass_through.E1 = [  1,0;
                     0,0;
                     0,0;
                     0,0;
                     0,0;
                     0,1];

pass_through.E2 = [  0,0;
                     0,0;
                     0,1;
                     1,0;
                     0,0;
                     0,0];

pass_through.state_order = {'a1';'d1';'rho1';'a2';'d2';'rho2'};

pass_through.F = [0,E,E,E;E,0,E,E;E,E,0,E;E,E,E,0];
pass_through.H = [0,T,T,T;T,0,T,T;T,T,0,T;T,T,T,0];
pass_through.KL{1} = [   0,1,0,0,0,0;
                         0,0,1,0,0,0;];

pass_through.KL{2} = [   0,0,0,0,1,0;
                         0,0,0,0,0,1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
database = struct();
database.input = input;
database.output = output;
database.transfer_without = transfer_without;
database.center_without = center_without;
database.pass_through = pass_through;

end
```

Appendix E

MATLAB: 4-node Transportation System

```matlab
clear
close all

% Define all the variables
k = 15; nt = 21; nc = 16; nq = 14;

%Inflow at each end node. Parcels / unit of time
gamma_1 = 75/11.75;

%Outflow at each end node. Parcels / unit of time
phi_2 = 5; phi_3 = 5;

%Load speed of each truck. Parcels / unit of time
L_1 = 40; L_2 = 20; L_3 = 20;

% Unload speed of each truck. Parcels / unit of time
u_1 = 40; u_2 = 20; u_3 = 20;

%Travel times between nodes. tau_ij from node i to node j
tau_14 = 2; tau_41 = 2; tau_24 = 4; tau_42 = 4; tau_34 = 4; tau_43 = 4;

%Truck capacity
C_1 = 75; C_2 = 50; C_3 = 50;

T = Inf;
E = -Inf;

[A,B,C,D] = system_matrices_ABCD_No_Cf_Cs_cleaned(gamma_1,phi_2,phi_3,L_1
    ,L_2,L_3,u_1,u_2,u_3,tau_14,tau_41,tau_24,tau_42,tau_34,tau_43,C_1,C_2
    ,C_3);

```

```matlab

% === create structure matrices ===

S_A = double((A ~= -inf));
S_B = double((B ~= inf));
S_D = double((D ~= 0));

S_s = S_A * S_B * S_D;
G = digraph(transpose(S_s));
hasCycle = ~isdag(G);

% === check solvability ===
if hasCycle
    warning('There are cycles in the communications graph.');
else
    disp('There are no cycles in the communications graph.');
end


% === compute execution order of the implicit states ===
cycles = allcycles(G);
executionOrder = toposort(G);
inDegree = indegree(G);
levels = zeros(size(executionOrder));

% Assign nodes to levels
for i = 1:length(executionOrder)
    node = executionOrder(i);
    predNodes = predecessors(G, node); % Get parent nodes
    if isempty(predNodes)
        levels(i) = 1;
    else
        levels(i) = max(levels(ismember(executionOrder, predNodes))) + 1;
    end
end

% Group nodes by levels
maxLevel = max(levels);
executionCell = cell(maxLevel, 1);
for lvl = 1:maxLevel
    group = executionOrder(levels == lvl);
    fprintf('Execution Group %d: %s\n', lvl, num2str(group));
    executionCell{lvl} = group;
end



% === check for time invariance ===
%%
C_11 = C(1:83, 1:37);
D_11 = D(1:83, 1:37);

C_21 = C(84:end, 1:37);
```

```
D_21 = D(84:end, 1:37);


CD_11 = [C_11, D_11];
CD_21 = [C_21, D_21];

sum_time_time_invariance = sum(CD_11,2);
sum_quantitity_time_invariance = sum(CD_21,2);

if sum(sum_time_time_invariance) == length(sum_time_time_invariance) &&
    sum(sum_quantitity_time_invariance) == 0
     disp('The system is Time Invariant');
else
     warning('Warning: The system is Not Time Invariant!');
end



% === simulate the system ===


eig_vec1 = [0;2;2;3.875;7.875;7.875;7.875;7.875;7.875;7.875;1.875;
3.875;3.875;9.75;9.75;9.75;9.75;10.375;10.375;10.375;10.375;9.75;
9.125;10.375;10.375;9.75;9.125;10.375;10.375;9.125;10.375;
9.125;10.375;9.125;10.375;9.125;10.375;0;-125000;
-125000;10;0;0;75;37.5;37.5;1;37.5;37.5;37.5;37.5]';


eig_vec2= [37.37625;39.37625;39.37625;41.25125;44.7525;45.75;
44.7525;45.75;44.7525;45.75;39.25125;40.7525;41.75;47.12625;47.12625;
47.12625;47.12625;46.0025;47.2525;48.25;47;47.12625;47;47.2525;48.25;
47.12625;47;47.2525;48.25;46.0025;48.25;46.0025;48.25;47;47.2525;
47;47.2525;0;-224750;-25250;100;0;0;75;47.475;27.525;0;47.475;
47.475;27.525;27.525];

x_state = zeros(nt+nc+nq,k);
x_state(:,1) = eig_vec1;

if C_1 > C_2 + C_3
     error("Capacity of truck 1 is to large. It will bring to many goods
         for the rest to take. The model" + ...
          "will fail in this situation. Make sure: C_1 < C_2 + C_3")
end


for i = 2:k

     for level = 1:length(executionCell)
          rowsToCompute = executionCell{level};
          x_intermediate = maxplus(A,minplus(B,(C*x_state(:,i-1)+ D*x_state
              (:,i))));
          x_state(rowsToCompute,i) = x_intermediate(rowsToCompute);
     end
```

```
end
```

```
function C = maxplus(A, B)
    [m, n] = size(A);
    [n2, p] = size(B);
    if n ~= n2
        error('Matrix dimensions must match for multiplication');
    end
    C = -inf(m, p); % Initialize with - infinity
    for i = 1:m
        for j = 1:p
            C(i, j) = max(A(i, :) + B(:, j)');
        end
    end
end
```

```
function C = minplus(A,B)
T = inf;
ar = size(A,1);
ac = size(A,2);
br = size(B,1);
bc = size(B,2);
C = T.*ones(ar,bc);
if ac==br
for i= 1:ar
    for j= 1:bc
        for k = 1: ac
        C(i,j)= min(C(i,j),A(i,k)+B(k,j));
        end
    end
end
else
    disp('minplus matrix multiplication not possible')
end
end
```

```
function [A,B,C,D] = system_matrices_ABCD_No_Cf_Cs_cleaned(gamma_1,phi_2,
    phi_3,L_1,L_2,L_3,u_1,u_2,u_3,tau_14,tau_41,tau_24,tau_42,tau_34,
    tau_43,C_1,C_2,C_3)

% === initialiting the matrices ===
T = Inf;
E = -Inf;

D_time = zeros(90,51);
C_time = zeros(90,51);
D_quantity = zeros(24,51);
C_quantity = zeros(24,51);
B = T * ones(97,114);
A  = E * ones(51,97);



```

```
% === D time states ===

%arrival formula
D_time(4,11) = 1;
D_time(5,12) = 1;
D_time(6,13) = 1;

%arrival first and last
D_time(7,5) = 1;
D_time(8,6) = 1;
D_time(9,4) = 1;


%departure formulas
%d1
D_time(10,1) = 1; % second part min d1
D_time(11,1) = L_1 / (L_1 - gamma_1);

%d2
D_time(12,2) = 1;

%d3
D_time(13,3) = 1;

%d41
D_time(14,15) = 1;
D_time(15,16) = 1;


%d42
D_time(16,17) = 1;
D_time(16,38) = 1;
D_time(17,18) = 1;
D_time(17,39) = 1;
D_time(18,19) = 1;
D_time(18,40) = 1;


%d43
D_time(19,17) = 1;
D_time(19,38) = 1;
D_time(20,20) = 1;
D_time(20,39) = 1;
D_time(21,21) = 1;
D_time(21,40) = 1;


% d4.1
D_time(22,44) = 1/(L_2+L_3);
D_time(22,10) = (-L_2/(L_2+L_3))+1;
D_time(22,9) = L_2/(L_2+L_3);
D_time(22,7) = 1e8;
D_time(22,5) = -1e8;
```

```
D_time(23,44) = 1/(u_1+L_3);
D_time(23,10) = (-u_1/(u_1+L_3))+1;
D_time(23,9) = u_1/(u_1+L_3);
D_time(23,7) = 1e8;
D_time(23,5) = -1e8;
D_time(24,44) = 1/(u_1+L_2);
D_time(24,10) = (-L_2/(u_1+L_2))+1;
D_time(24,9) = L_2/(u_1+L_2);
D_time(24,7) = 1e8;
D_time(24,5) = -1e8;
D_time(25,44) = 1/(2*u_1);
D_time(25,10) = (-u_1/(2*u_1)) + 1;
D_time(25,9) = u_1/(2*u_1);
D_time(26,44) = 1/(L_2+L_3);
D_time(26,10) = (-L_3/(L_2+L_3))+1;
D_time(26,9) = L_3/(L_2+L_3);
D_time(26,7) = 1e8;
D_time(26,6) = -1e8;
D_time(27,44) = 1/(u_1+L_3);
D_time(27,10) = (-L_3/(u_1+ L_3))+1;
D_time(27,9) = L_3/(u_1+L_3);
D_time(27,7) = 1e8;
D_time(27,6) = -1e8;
D_time(28,44) = 1/(u_1+L_2);
D_time(28,10) = (-u_1/(L_2+u_1))+1;
D_time(28,9) = u_1/(L_2+u_1);
D_time(28,7) = 1e8;
D_time(28,6) = -1e8;

%d422
D_time(29,10) = 1;
D_time(29,5) = 1e8;
D_time(29,8) = -1e8;
D_time(30,10) = 1;
D_time(30,5) = 1e8;
D_time(30,8) = -1e8;
D_time(31,9) = 1;
D_time(31,44) = 1/u_1;
D_time(31,7) = 1e8;
D_time(31,5) = -1e8;
D_time(32,9) = 1;
D_time(32,44) = 1/L_2;
D_time(32,7) = 1e8;
D_time(32,5) = -1e8;

%d423
D_time(33,9) = 1;
D_time(33,7) = 1e8;
D_time(33,5) = -1e8;
D_time(34,9) = 1;
D_time(34,44) = 1/L_2;
D_time(34,7) = 1e8;
D_time(34,5) = -1e8;
```

```
D_time(35,9) = 1;
D_time(35,7) = 1e8;
D_time(35,5) = -1e8;
D_time(36,9) = 1;
D_time(36,44) = 1/u_1;
D_time(36,7) = 1e8;
D_time(36,5) = -1e8;
D_time(37,10) = 1;
D_time(37,5) = 1e8;
D_time(37,8) = -1e8;
D_time(38,10) = 1;
D_time(38,44) = 1/L_2;
D_time(38,5) = 1e8;
D_time(38,8) = -1e8;
D_time(39,10) = 1;
D_time(39,44) = 1/u_1;
D_time(39,5) = 1e8;
D_time(39,8) = -1e8;


%d432
D_time(40,10) = 1;
D_time(40,6) = 1e8;
D_time(40,8) = -1e8;
D_time(41,10) = 1;
D_time(41,6) = 1e8;
D_time(41,8) = -1e8;
D_time(42,9) = 1;
D_time(42,44) = 1/u_1;
D_time(42,7) = 1e8;
D_time(42,6) = -1e8;
D_time(43,9) = 1;
D_time(43,44) = 1/L_3;
D_time(43,7) = 1e8;
D_time(43,6) = -1e8;

%d433
D_time(44,9) = 1;
D_time(44,7) = 1e8;
D_time(44,6) = -1e8;
D_time(45,9) = 1;
D_time(45,44) = 1/L_3;
D_time(45,7) = 1e8;
D_time(45,6) = -1e8;
D_time(46,9) = 1;
D_time(46,7) = 1e8;
D_time(46,6) = -1e8;
D_time(47,9) = 1;
D_time(47,44) = 1/u_1;
D_time(47,7) = 1e8;
D_time(47,6) = -1e8;
D_time(48,10) = 1;
D_time(48,6) = 1e8;
```

```
D_time(48,8) = -1e8;
D_time(49,10) = 1;
D_time(49,44) = 1/L_3;
D_time(49,6) = 1e8;
D_time(49,8) = -1e8;
D_time(50,10) = 1;
D_time(50,44) = 1/u_1;
D_time(50,6) = 1e8;
D_time(50,8) = -1e8;


% === conditions steps (f) ===

D_time(51,44) = 1/ (L_2 + L_3);
D_time(51,10) = (-L_2/(L_2+L_3))+1;
D_time(51,9) = L_2/(L_2+L_3) ;
D_time(52,44) = 1/ (u_1 + L_3);
D_time(52,10) = (-u_1/(u_1+L_3))+1;
D_time(52,9) = u_1/(u_1+L_3) ;
D_time(53,44) = 1/ (2 * u_1);
D_time(53,10) = (-u_1/(2 * u_1))+1;
D_time(53,9) = u_1/(2 * u_1);
D_time(54,44) = 1/ (L_2 + u_1);
D_time(54,10) = (-L_2/(L_2+u_1)) + 1;
D_time(54,9) = L_2/(L_2+u_1) ;
D_time(55,44) = 1/L_3;
D_time(55,10) = 1;
D_time(56,44) = 1/u_1;
D_time(56,10) = 1;
D_time(57,9) = 1;
D_time(58,9) = 1;
D_time(59,10) = 1;
D_time(60,10) = 1;
D_time(61,44) = 1/ (L_2 + L_3);
D_time(61,10) = (-L_3/(L_2+L_3)) + 1;
D_time(61,9) = L_3/(L_2+L_3) ;
D_time(62,44) = 1/ (u_1 + L_2);
D_time(62,10) = (-u_1/(u_1+L_2)) + 1;
D_time(62,9) = u_1/(u_1+L_2);
D_time(63,44) = 1/ (2 * u_1);
D_time(63,10) = (-u_1/(2 * u_1)) + 1;
D_time(63,9) = u_1/(2 * u_1);
D_time(64,44) = 1/ (L_3 + u_1);
D_time(64,10) = (-L_3/(L_3+u_1)) + 1;
D_time(64,9) = L_3/(L_3+u_1);
D_time(65,44) = 1/L_2;
D_time(65,10) = 1;
D_time(66,9) = 1;
D_time(67,10) = 1;
D_time(68,9) = 1;
D_time(68,44) = 1/u_1;
D_time(69,9) = 1;
D_time(69,44) = 1/L_2;
```

```
D_time(70,10) = 1;
D_time(71,10) = 1;
D_time(72,9) = 1;
D_time(72,44) = 1/L_3;
D_time(73,10) = 1;
D_time(74,10) = 1;
D_time(74,44) = 1/u_1;
D_time(75,10) = 1;
D_time(75,44) = 1/L_2;
D_time(76,9) = 1;
D_time(77,9) = 1;
D_time(78,10) = 1;
D_time(78,44) = 1/L_3;
D_time(79,9) = 1;

%condition1
D_time(80,22) = 1e5;
D_time(80,23) = -1e5;
D_time(80,7) = 1e8;
D_time(80,5) = -1e8;
D_time(81,22) = -1e5;
D_time(81,24) = 1e5;
D_time(81,7) = 1e8;
D_time(81,5) = -1e8;
D_time(82,22) = -1e5;
D_time(82,25) = 1e5;
D_time(82,7) = 1e8;
D_time(82,5) = -1e8;
D_time(84,26) = 1e5;
D_time(84,27) = -1e5;
D_time(84,7) = 1e8;
D_time(84,6) = -1e8;
D_time(85,26) = -1e5;
D_time(85,28) = 1e5;
D_time(85,7) = 1e8;
D_time(85,6) = -1e8;
D_time(86,26) = -1e5;
D_time(86,29) = 1e5;
D_time(86,7) = 1e8;
D_time(86,6) = -1e8;

%condition2
D_time(87,30) = 1e5;
D_time(87,31) = -1e5;
D_time(87,7) = 1e8;
D_time(87,5) = -1e8;
D_time(88,32) = 1e5;
D_time(88,33) = -1e5;
D_time(88,7) = 1e8;
D_time(88,6) = -1e8;

%condition3

```

```
D_time(89,34) = 1e5;
D_time(89,35) = -1e5;
D_time(89,5) = 1e8;
D_time(89,8) = -1e8;
D_time(90,36) = 1e5;
D_time(90,37) = -1e5;
D_time(90,6) = 1e8;
D_time(90,8) = -1e8;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%stack at departure formulas
D_quantity(1,11) = gamma_1 - L_1;
D_quantity(1,1) = L_1;
D_quantity(2,12) = u_2-phi_2;
D_quantity(2,2) = -u_2;
D_quantity(3,13) = u_3-phi_3;
D_quantity(3,3) = -u_3;

%rho1
D_quantity(4,11) = L_1;
D_quantity(4,1) = -L_1;
%rho42
D_quantity(5,15) = L_2;
D_quantity(5,5) = -L_2;
D_quantity(6,15) = L_2;
D_quantity(6,4) = -L_2;
D_quantity(7,15) = u_1;
D_quantity(7,5) = -u_1;
D_quantity(8,15) = u_1;
D_quantity(8,4) = -u_1;


%rho43
D_quantity(9,16) = L_3;
D_quantity(9,6) = -L_3;
D_quantity(10,16) = L_3;
D_quantity(10,4) = -L_3;
D_quantity(11,16) = u_1;
D_quantity(11,6) = -u_1;
D_quantity(12,16) = u_1;
D_quantity(12,4) = -u_1;

%delta1
%row 13 is left emty
D_quantity(14,7) = -1e5;
D_quantity(14,8) = 1e5;
D_quantity(15,7) = 1e5;
D_quantity(15,8) = -1e5;

%row 16 is left empty for s

```

```
%rho comp2
D_quantity(17,44) = 1;
D_quantity(17,46) = -1;
D_quantity(18,44) = 1;
D_quantity(18,46) = -1;
D_quantity(18,47) = C_3 - C_2;

%rho real2
D_quantity(19,45) = 1;
D_quantity(20,48) = 1;

%rho comp3
D_quantity(21,44) = 1;
D_quantity(21,45) = -1;
D_quantity(22,44) = 1;
D_quantity(22,45) = -1;
D_quantity(22,47) = C_2 - C_3;

%rho real3
D_quantity(23,46) = 1;
D_quantity(24,50) = 1;

D = [D_time; D_quantity];

D(83,7) = 1e7;
D(83,5) = -1e7;
row_90_new = zeros(1, size(D, 2));
D = [D(1:89, :); row_90_new; D(90:end, :)];
D(90,34) = 1e5;
D(90,35) = -1e5;
D(90,5) = 1e8;
D(90,8) = -1e8;
D(90,47) = -1e8 * (C_3-C_2);
row_92_new = zeros(1, size(D, 2));
D = [D(1:91, :); row_92_new; D(92:end, :)];
D(92,36) = 1e5;
D(92,37) = -1e5;
D(92,6) = 1e8;
D(92,8) = -1e8;
D(92,47) = -1e8 * (C_2 - C_3);
row_87_new = zeros(1, size(D, 2));
D = [D(1:86, :); row_87_new; D(87:end, :)];
D(87,7) = 1e7;
D(87,6) = -1e7;
row_41_new = zeros(1, size(D, 2));
D = [D(1:40, :); row_41_new; D(41:end, :)];
row_43_new = zeros(1, size(D, 2));
D = [D(1:42, :); row_43_new; D(43:end, :)];
D(41,10) = 1;
D(41,6) = 1e8;
D(41,8) = -1e8;
D(41,47) = (C_2- C_3) * 1e5;
D(43,10) = 1;
```

```
D(43,6) = 1e8;
D(43,8) = -1e8;
D(43,47) = (C_2- C_3) * 1e5;
row_30_new = zeros(1, size(D, 2));
D = [D(1:29, :); row_30_new; D(30:end, :)];
row_32_new = zeros(1, size(D, 2));
D = [D(1:31, :); row_32_new; D(32:end, :)];
D(30,10) = 1;
D(30,5) = 1e8;
D(30,8) = -1e8;
D(30,47) = (C_3- C_2) * 1e5;
D(32,10) = 1;
D(32,5) = 1e8;
D(32,8) = -1e8;
D(32,47) = (C_3- C_2) * 1e5;


% === Create C matrix ===
C_time(1, 14) = 1;
C_time(2,15) = 1;
C_time(3,16) = 1;

%d1
C_time(11,41) = 1/(L_1-gamma_1);
C_time(11,11) = -gamma_1 / (L_1 - gamma_1);


%d2
C_time(12,49) = 1/u_2;

%d22
C_time(13,51) = 1/u_3;

%s1
C_quantity(1,41) = 1;
C_quantity(1,11) = -gamma_1;

%s2
C_quantity(2,42) = 1;
C_quantity(2,12) = phi_2;

%s3
C_quantity(3, 43) = 1;
C_quantity(3,13) = phi_3;

C = [C_time; C_quantity];
row_90_new = zeros(1, size(C, 2));
C = [C(1:89, :); row_90_new; C(90:end, :)];
row_92_new = zeros(1, size(C, 2));
C = [C(1:91, :); row_92_new; C(92:end, :)];
row_87_new = zeros(1, size(C, 2));
C = [C(1:86, :); row_87_new; C(87:end, :)];
row_43_new = zeros(1, size(C, 2));
```

```
C = [C(1:42, :); row_43_new; C(43:end, :)];
row_41_new = zeros(1, size(C, 2));
C = [C(1:40, :); row_41_new; C(41:end, :)];
row_32_new = zeros(1, size(C, 2));
C = [C(1:31, :); row_32_new; C(32:end, :)];
row_30_new = zeros(1, size(C, 2));
C = [C(1:29, :); row_30_new; C(30:end, :)];



% === Create B matrix ===

for i = [1,2,3,4,5,6]
    B(i, i) = 0;
end

%af
B(7,7) = 0;
B(7,8) = 0;

%as
B(8,7) = 0;
B(9,8) = 0;

%lf
B(10,9) = 0;
B(11,7) = 0;
B(11,8) = 0;

%ls
%uses that of af,as,ls

%d1
B(12,10) = C_1/L_1;
B(12,11) = 0;
%d2
B(13,12) = 0;

%d3
B(14,13) = 0;

%d41
B(15,14) = 0;
B(16,15) = 0;

%d42
B(17,16) = 0;
B(18,17) = 0;
B(19,18) = 0;

%d43
B(20,19) = 0;
B(21,20) = 0;
```

```
B(22,21) = 0;

%d4.1
B(23,22) = 0;
B(24,23) = 0;
B(25,24) = 0;
B(26,25) = 0;
B(27,26) = 0;
B(28,27) = 0;
B(29,28) = 0;

%d422
B(30,29) = (C_2/u_1) ;%- (C_2 * 1e5);
B(31,30) = (C_2/L_2) ;%- (C_2 * 1e5);
B(32,31) = -C_3/u_1;
B(33,32) = -C_3/L_2;

%d423
B(34,33) = C_2/L_2;
B(34,34) = 0;
B(35,35) = C_2/u_1;
B(35,36) = 0;
B(36,37) = 0;
B(37,38) = -C_3/L_2;
B(38,39) = -C_3/u_1;

%d432
B(39,40) = C_3/u_1 ; %- (C_3 * 1e5);
B(40,41) = C_3/L_3 ; %- (C_3 * 1e5);
B(41,42) = -C_2/u_1;
B(42,43) = -C_2/L_3;

%d433
B(43,44) = C_3/L_3;
B(43,45) = 0;
B(44,46) = C_3/u_1;
B(44,47) = 0;
B(45,48) = 0;
B(46,49) = -C_2/L_3;
B(47,50) = -C_2/u_1;

%c1 parts
B(48,51) = 0;
B(49,52) = 0;
B(50,53) = 0;
B(51,54) = 0;
B(52,55) = 0;
B(53,56) = 0;
B(54,57) = 0;
B(55,58) = 0;
B(56,59) = 0;
B(57,60) = 0;
B(58,61) = 0;
```

```
B(59,62) = 0;
B(60,63) = 0;
B(61,64) = 0;
B(62,65) = 0;
B(63,66) = 0;
B(64,67) = 0;

%c2 parts
B(65,68) = 0;
B(66,69) = 0;
B(67,70) = 0;
B(68,71) = 0;
B(69,72) = 0;
B(70,73) = 0;

%c3 parts
B(71,74) = 0;
B(72,75) = 0;
B(73,76) = 0;
B(74,77) = 0;
B(75,78) = 0;
B(76,79) = 0;

%c1
B(77,80) = 0;
B(77,81) = 0;
B(77,82) = 0;
B(77,83) = 0;

B(78,84) = 0;
B(78,85) = 0;
B(78,86) = 0;

%c2
B(79,87) = 0;
B(79,83) = 0;

B(80,88) = 0;

%c3
B(81,89) = 0;

B(82,90) = 0;
B(82,83) = 0;

%s1
B(83,91) = 0;

%s2
B(84,92) = 0;
B(85,106) = 0;

%s3
```

```
B(86,93) = 0;

%rho1
B(87,94) = 0;

%rho42
B(88,95) = 0;
B(88,96) = 0;
B(88,97) = 0;
B(88,98) = 0;

%rho43
B(89,99) = 0;
B(89,100) = 0;
B(89,101) = 0;
B(89,102) = 0;

%delta1
B(90,103) = 0;
B(91,104) = 1;
B(91,105) = 1;

new_col_90 = T * ones(size(B, 1), 1);
B = [B(:, 1:89), new_col_90, B(:, 90:end)];
B(81,90) = 0;
new_col_92 = T * ones(size(B, 1), 1);
B = [B(:, 1:91), new_col_92, B(:, 92:end)];
B(82,92) = 0;
new_col_87 = T * ones(size(B, 1), 1);
B = [B(:, 1:86), new_col_87, B(:, 87:end)];
B(78,87) = 0;
B(80,87) = 0;
B(81,87) = 0;
new_col_41 = T * ones(size(B, 1), 1);
B = [B(:, 1:40), new_col_41, B(:, 41:end)];
new_col_43 = T * ones(size(B, 1), 1);
B = [B(:, 1:42), new_col_43, B(:, 43:end)];
B(39,41) = C_3/u_1 ;
B(40,43) = C_3/L_3 ;
new_col_30 = T * ones(size(B, 1), 1);
B = [B(:, 1:29), new_col_30, B(:, 30:end)];
new_col_32 = T * ones(size(B, 1), 1);
B = [B(:, 1:31), new_col_32, B(:, 32:end)];
B(30,30) = (C_2/u_1) ;
B(31,32) = (C_2/L_2) ;
B(92,114) = 0;
B(93,115) = 0;
B(94,116) = 0;
B(94,117) = 0;
B(95,118) = 0;
B(96,119) = 0;
B(97,120) = 0;
B(97,121) = 0;
```

```
% === create A matrix ===

A(1,1) = tau_41;
A(2,2) = tau_42;
A(3,3) = tau_43;
A(4,4) = tau_14;
A(5,5) = tau_24;
A(6,6) = tau_34;
%af
A(7,7) = 0;
%as
A(8,8) = 0;
A(8,9) = 0;

%lf
A(9,10) = 0;
A(9,11) = 0;
%ls
A(10,8) = 0;
A(10,9) = 0;
A(10,10) = 0;
%d1
A(11,12) = 0;

%d2
A(12,13) = 0;

%d3
A(13,14) = 0;

%d41
A(14,15) = 0;
A(14,16) = 0;

%d42
A(15,17) = 0;
A(15,18) = 0;
A(15,19) = 0;

%d43
A(16,20) = 0;
A(16,21) = 0;
A(16,22) = 0;

%d4.1
A(17,23) = 0;
A(17,24) = 0;
A(17,25) = 0;
A(17,26) = 0;
A(17,27) = 0;
A(17,28) = 0;
A(17,29) = 0;
```

```
%d422
A(18,30) = 0;
A(18,31) = 0;
A(18,32) = 0;
A(18,33) = 0;

%d423
A(19,34) = 0;
A(19,35) = 0;
A(19,36) = 0;
A(19,37) = 0;
A(19,38) = 0;

%d432
A(20,39) = 0;
A(20,40) = 0;
A(20,41) = 0;
A(20,42) = 0;

%d433
A(21,43) = 0;
A(21,44) = 0;
A(21,45) = 0;
A(21,46) = 0;
A(21,47) = 0;

%conditions
%c1 parts
A(22,48) = 0;
A(22,49) = 0;
A(22,50) = 0;
A(22,51) = 0;

A(23,52) = -C_2/L_3;
A(23,53) = -C_2/u_1;
A(24,54) = C_2/L_2;
A(24,55) = C_2/u_1;
A(25,56) = C_3/L_3;
A(25,57) = C_3/u_1;
A(26,58) = 0;
A(26,59) = 0;
A(26,60) = 0;
A(26,61) = 0;
A(27,62) = -C_3/L_2;
A(27,53) = -C_3/u_1;
A(28,63) = C_3/L_3;
A(28,55) = C_3/u_1;
A(29,64) = C_2/L_2;
A(29,57) = C_2/u_1;

%c2 parts
A(30,65) = -C_3/u_1;
```

```
A(30,66) = -C_3/L_2;
A(31,67) = C_3/u_1;
A(31,68) = C_3/L_3;
A(32,65) = -C_2/u_1;
A(32,69) = -C_2/L_3;
A(33,67) = C_2/u_1;
A(33,70) = C_2/L_2;

%c3 parts
A(34,71) = -C_3/u_1;
A(34,72) = -C_3/L_2;
A(35,73) = C_3/u_1;
A(35,74) = C_3/L_3;
A(36,71) = -C_2/u_1;
A(36,75) = -C_2/L_3;
A(37,73) = C_2/u_1;
A(37,76) = C_2/L_2;

%c1
A(38,77) = 0;
A(38,78) = 0;

%c2
A(39,79) = 0;
A(39,80) = 0;

%c3
A(40,81) = 0;
A(40,82) = 0;

%stacks
A(41,83) = 0;
A(42,84) = 0;
A(42,85) = 0;
A(43,85) = 0;
A(43,86) = 0;

%truck loads
A(44,87) = 0;
A(45,88) = 0;
A(46,89) = 0;

%delta1
A(47,90) = 0;
A(47,91) = 0;
A(48,92) = 0;
A(48,93) = 0;
A(49,94) = 0;
A(50,95) = 0;
A(50,96) = 0;
A(51,97) = 0;
```

References

[1] B. Heidergott, G. J. Olsder, and J. van der Woude, Eds., *Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications* (Princeton Series in Applied Mathematics), eng. Princeton: Princeton University Press, 2014.

[2] B. D. Schutter and T. v. d. Boom, "Max-plus algebra and max-plus linear discrete event systems: An introduction," in *2008 9th International Workshop on Discrete Event Systems*, May 2008, pp. 36–42.

[3] T. v. d. Boom, A. Gupta, B. D. Schutter, and R. Beek, "Max-Min-Plus-Scaling Systems in a Discrete-Event Framework with an Application in Urban Railway," *IFAC-PapersOnLine*, 22nd IFAC World Congress, vol. 56, no. 2, pp. 7906–7911, Jan. 2023.

[4] S. Markkassery, T. v. d. Boom, and B. D. Schutter, "Eigenvalues of Time-invariant Max-Min-Plus-Scaling Discrete-Event Systems," in *2024 European Control Conference (ECC)*, Jun. 2024, pp. 2017–2022.

[5] S. Markkassery, T. v. d. Boom, and B. D. Schutter, "Dynamics of Implicit Time-Invariant Max-Min-Plus-Scaling Discrete-Event Systems," en, 2025, To be Published.

[6] S. Markkassery, T. v. d. Boom, and B. D. Schutter, "Stability of Time-invariant Max-Min-Plus-Scaling Discrete-Event Systems with Diverse States," *IFAC-PapersOnLine*, 17th IFAC Workshop on discrete Event Systems WODES 2024, vol. 58, no. 1, pp. 60–65, Jan. 2024.

[7] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*, en, ISBN: 9781139061759 Publisher: Cambridge University Press, Jun. 2017.

[8] S. P. Boyd and L. Vandenberghe, *Convex optimization*, en, Version 29. Cambridge New York Melbourne New Delhi Singapore: Cambridge University Press, 2023.

[9] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness* (A series of books in the mathematical sciences), eng, 27. print. New York [u.a]: Freeman, 2009.

[10] T. v. d. Boom and B. D. Schutter, *Optimization for Systems and Control* (Lecture Notes for the course SC42056). Delft Center for Systems and Control Delft University of Technology Mekelweg 2, 2628 CD Delft, The Netherlands, Sep. 2022.

[11] S. R. Daams, “Control Strategies for Max-Min-Plus-Scaling Systems,” en, 2024.

[12] G. Olsder, “On structural properties of min-max systems,” springer, Jun. 1994, pp. 237–246.

[13] G. J. Olsder, “On min-max-plus systems, nonexpansive mappings and periodic solutions,” en, Jan. 2001.

[14] S. A. Rosyada, Siswanto, and V. Y. Kurniawan, “Bases in Min-Plus Algebra,” en, ISSN: 2352-5398, Atlantis Press, Nov. 2021, pp. 313–316.

[15] Y. Cheng, “A Survey of the Theory of Min-Max Systems,” en, in *Lecture Notes in Computer Science*, ISSN: 0302-9743, 1611-3349, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 616–625.

[16] D. C. Lay, S. R. Lay, and J. McDonald, *Linear algebra and its applications*, Fifth edition. Boston: Pearson, 2016.

[17] V. M. van Heijningen, “Solving solvability of implicit max-min-plus-scaling systems,” en, 2025.

[18] J. van Drunen, M. Cooper, J. Wempe, and M. van Gils, “Optimizing logistic systems using max-min-plus scaling in ABCD canonical form,” en, Jun. 2025.

[19] T. v. d. Boom, S. Markkassery, and B. D. Schutter, “Bounds on the growth rate of time-invariant switching max-min-plus-scaling discrete-event systems,” *IFAC-PapersOnLine*, 26th International Symposium on Mathematical Theory of Networks and Systems MTNS 2024, vol. 58, no. 17, pp. 404–409, Jan. 2024.

[20] B. D. Schutter and M. Heemels, *Modeling and Control of Hybrid Systems* (Lecture Notes for the course SC42075). Delft, The Netherlands: Delft Center for Systems and Control, Delft University of Technology en Hybrid & Networked Systems group, Eindhoven University of Technology, 2021.

Glossary

List of Acronyms

DE	Discrete Event
DES	Discrete Event System
LPP	Linear Programming Problem
MILP	Mixed-Integer Linear Programming
MMPS	Max-Min-Plus-Scaling
S-MMPS	Switching Max-Min-Plus-Scaling
MMP	Max-Min-Plus
URS	Urban Railway System
VRP	Vehicle Routing Problem
ME	Mechanical Engineering
TPS	Transportation System

List of Symbols

$\boxtimes$	Kronecker product operator
λ	Additive eigenvalue or growth rate
$\mathbb{N}$	Set of positive integers including 0
$\mathbb{P}$	Max-plus Hilbert projective norm
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_{\epsilon}$	Set of real numbers including ϵ
$\mathbb{R}_{\top}$	Set of real numbers including $\top$
$\mathbb{R}_{c}$	Set of real numbers including ϵ and $\top$
$\mathbb{Z}$	Set of integers
$\mathbb{Z}^{+}$	Set of positive integers

μ	Multiplicative eigenvalue or periodic point
$\oplus$	Max-plus addition operator ('o-plus')
$\oplus'$	Min-plus addition operator ('o-plus-prime')
$\otimes$	Max-plus multiplication operator ('o-times')
$\otimes'$	Min-plus multiplication operator ('o-times-prime')
$\top$	Min-plus zero element $\top = \infty$
ε	Max-plus zero element $\varepsilon = -\infty$
e	Max-plus and min-plus one element $e = 0$
M	A sufficiently large number
n_q	Number of quantity states
n_t	Number of time states
v	Additive or multiplicative eigenvector or fixed point