Predicting Processing Time of Stochastic Switching Max-Plus Systems

Samuel Hoogerwerf

Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Predicting Processing Time of Stochastic Switching Max-Plus Systems

Thesis

Samuel Hoogerwerf

May 30, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The goal in this thesis is to make a prediction on the total processing time for logistical systems to complete their tasks. For simple systems that can be described using a regular max plus state space model this is done by calculating the systems eigenvalue and multiplying that by the number of iterations required. But, for more complex systems that can only be described using a switching max-plus state space model such as a production line that can change during production, or a rail network where some rail segments become unavailable at times. If these changes to the system are fully within control of the operator they are easily accounted for when predicting the total processing time. But, if the system switches stochastically then one would need to simulate all possible permutations of operation to know the average processing time of the system. Alternatively, one might approximate this average processing time using some simplified model.

We found three main methods to predict the total processing time. Firstly, we can fit a generalized extreme value distribution to a histogram of the systems performance and then extrapolate this distribution function. Secondly, we can fit a marginal cost model to a limited simulation of the system and then extrapolate based on that model. Lastly, we can rewrite the expectation of how long the system might take to go through N iterations to an inequality by adding an arbitrary diagonal matrix S, which if minimized can be used to approximate total processing time.

All three prediction methods can, when fitted properly predict at least within 15% accuracy with the fitted extreme value distribution generally performing best. We also found that it is possible to save total processing time using a controller based on the marginal cost model if it is possible to exert some limited control over the mode switching process.

Table of Contents

Chapter 1

Introduction

Complex logistical systems can be modeled using max-plus algebra. This is a system of algebra with slightly altered rules the maximum function is used in the place of addition and addition in place of multiplication. Using this system of algebra it is possible to compactly describe simple logistical systems in state space form. An example of such a simple system might be a factory line that always produces the same product, or a public transport system that never changes it's schedule.

But then, what if the factory can produce a variety of products by mixing production lines over the course of the day. Or, what if you want to model the public transport system including possible backup lines if the ones in the official schedule fall through due to accidents or other circumstances. In these cases can also be described using max-plus algebra by using a switching state space model. In the case of the factory the owner presumably has full control over what is produced when and on which lanes, this we might call controlled switching. On the other hand accidents could hardly be planned, but the odds of them happening can be approximated. In such a case we would talk about stochastic switching.

1-1 Problem Definition

With simple non switching max-plus linear systems it is trivially simple to calculate total processing time, the same can be said for Switching Max-Plus-Linear (SMPL) systems that are fully controllable. The difficulty is found in stochastic SMPL systems, as to predict the total processing time for these systems requires simulating every permutation of package order and calculating the odds for that particular permutation to occur. As the number of possible permutations grow exponentially for every linear increase in prediction horizon this method quickly becomes untenable. As such the main question we want to answer is:

Can we approximate the mean total processing time over N iterations for a stochastic SMPL system? At the very least the difficulty of calculating this approximation should scale sub-exponentially with increasing N while being at most 10% off the actual mean processing time.

To calculate such an approximation we need a simplified model of the system that will allow extrapolation without increasing the calculation load. This simplified model should only consider the stochastic distribution of the possible operation modes. It should not require a list of all upcoming modes. In our example that would be a list of packages with incoming lane and outgoing lane included.

Secondly if we were to have such a model it might be possible to invert it and use it as a controller. Say in the case of the package sorter example we could manage a buffer of 5 packages before entering them in the machine and then every iteration pick the fastest to process package out of that buffer. Using this limited level of control, as we still don't control which package enters our buffer, we could reduce total processing time and thus process more packages per day.

Can we exert some limited control over a stochastic SMPL system by using buffered modes in some way that is faster than simulating all possible permutations of inputting the buffer?

1-2 Case Study

Throughout especially the second part of this thesis we will use an ongoing example of a very simple arbitrary stochastic SMPL system. We could have chosen anything ranging from a railway network to an automotive production line. We chose this arbitrary system as it is simple and useful when visualizing the mathematics since every package is an iteration and every possible package path is a mode.

Our example system can only be modeled using a stochastic SMPL model is the machine in Figure 1-1. Let's say this is a package sorting machine of some kind, clearly it can operate in four possible modes of operation:

1. A package comes in on lane 1 and needs to go to outgoing lane A.
2. A package comes in on lane 2 and needs to go to outgoing lane A.
3. A package comes in on lane 1 and needs to go to outgoing lane B.
4. A package comes in on lane 2 and needs to go to outgoing lane B.

Every single one of these modes of operation on their own can be described with a simple Max-Plus-Linear (MPL) model but we require switching MPL algebra to describe the system as a whole.

1-3 Outline

In the first few sections of Chapter 2 we go through how the basics of MPL algebra and how to write state space models. Then in the last few sections of Chapter 2 we extend this to SMPL systems and lay out the foundations for the actual research questions. With these basics resolved we can attempt to create some prediction models the first of which is Chapter 3 where we rewrite the approximation problem as an inequality. Another model is discussed

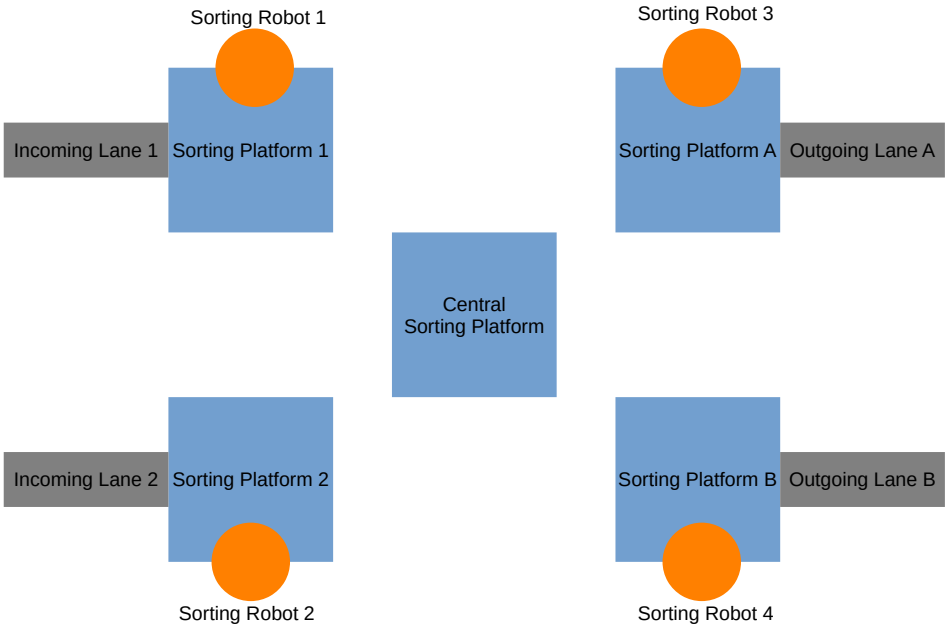

Figure 1-1: a top down overview of the sorting system

in Chapter 4 where we fit a marginal cost model that can be used for approximating total processing time. Within this chapter in Section 4-6 we discuss how a fitted marginal cost model can be used to reduce the total processing time for the same number of packages, if we presort the input. The third and last method of predicting total processing time is discussed in Chapter 5 where we fit extreme value distributions to a simulation of the system. Lastly, in Chapter 6 we run a large Monte-Carlo simulation to evaluate the relative accuracy of the different models.

Chapter 2

Max-Plus Basics

Max-Plus-Linear (MPL) algebra is based on a fundamentally different set of operations than regular Linear Algebra. As such we will explain all the basic operations and definitions in this chapter. These basics are discussed in many overview papers and books like [1][2][3][4][5].

2-1 Basic Definitions

In MPL algebra the basic operators of linear algebra are in a sense all shifted down one order. This Below we convert basic MPL expressions (left) to their conventional equivalent (right). The notation we use in this thesis as defined here can be found in [2].

$$a \oplus b = \max(a, b) \tag{2-1}$$
$$a \otimes b = a + b \tag{2-2}$$
$$a^{\otimes^b} = a \cdot b \tag{2-3}$$
$$\bigoplus_{i=1}^{n} a_i = max(a_1, a_2, \ldots, a_n) \tag{2-4}$$
$$\bigotimes_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n \tag{2-5}$$
$$\varepsilon = -\infty \tag{2-6}$$

Remark 2-1.1. Inverse Operations
A point to note is that $\oplus$ has no inverse.[2] There is no way to reverse engineer 2 out of $2 \oplus 5 = \max(2, 5) = 5$. Higher order MPL operations do have their respective inverses but these won't be used in this thesis.

These different operators are used on an extended spectrum of numbers relative to the set $\mathbb{R}$ in conventional algebra. With the defining difference being the inclusion of $-\infty$ which will be referred to in this paper as ε as is convention.

Definition 2-1.1. Extended Real Numbers
As such the set of real numbers used in regular algebra is increased in scope for MPL algebra as can be seen in Eq. (2-7)

$$\mathbb{R}_\varepsilon \stackrel{\mathbf{def}}{=} \mathbb{R} \cup -\infty = \mathbb{R} \cup \varepsilon \tag{2-7}$$

In MPL algebra ε functions effectively the same way 0 does in regular algebra. As has been discussed in Definition 2-4.2 ε is used in those A matrix elements where there is no corresponding edge in the graph.

$$a \otimes \varepsilon \stackrel{\mathbf{def}}{=} \varepsilon \otimes a \stackrel{\mathbf{def}}{=} \varepsilon \tag{2-8}$$

2-2 Matrix Operations

When only considering scalar operations one would be right to wonder what MPL algebra has to add other than some new notation. The answer to this question are the interesting parallels between conventional- and MPL algebra. We again used the notation defined in [2].

Definition 2-2.1. Matrix Addition
Paralleling conventional algebra for addition to take place, matrices need to be of the same size. Thus, for $A, B \in \mathbb{R}_\varepsilon^{m \times n}$ addition is extended in the following way:

$$(A \oplus B)_{ij} = a_{ij} \oplus b_{ij} = max(a_{ij}, b_{ij}) \tag{2-9}$$

Example 2-2.1. Matrix Addition
A Matrix MPL addition example.

$$\begin{bmatrix} 1 & 4 & 3 \\ 9 & 6 & 1 \\ 3 & 2 & 5 \end{bmatrix} \oplus \begin{bmatrix} 4 & 2 & 3 \\ 4 & 6 & 6 \\ 1 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 1 \oplus 4 & 4 \oplus 2 & 3 \oplus 3 \\ 9 \oplus 4 & 6 \oplus 6 & 1 \oplus 6 \\ 3 \oplus 1 & 2 \oplus 1 & 5 \oplus 3 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 3 \\ 9 & 6 & 6 \\ 3 & 2 & 5 \end{bmatrix} \tag{2-10}$$

Definition 2-2.2. Matrix Multiplication
Max-plus multiplication has the same requirements conventional matrix multiplication has: $A \in \mathbb{R}_\varepsilon^{m \times r}$ and $B \in \mathbb{R}_\varepsilon^{r \times n}$

$$(A \otimes B)_{ij} = \bigoplus_{k=1}^{r} A_{ik} \otimes B_{kj} \tag{2-11}$$

Example 2-2.2. Matrix Multiplication
An example of MPL multiplication.

$$\begin{bmatrix} 6 & \varepsilon & 2 \\ 4 & 5 & 7 \end{bmatrix} \otimes \begin{bmatrix} 2 & 7 \\ 7 & 4 \\ 2 & 9 \end{bmatrix} = \begin{bmatrix} 6 \otimes 2 \oplus \varepsilon \otimes 7 \oplus 2 \otimes 2 & 6 \otimes 7 \oplus \varepsilon \otimes 4 \oplus 2 \otimes 9 \\ 4 \otimes 2 \oplus 5 \otimes 7 \oplus 7 \otimes 2 & 4 \otimes 7 \oplus 5 \otimes 4 \oplus 7 \otimes 9 \end{bmatrix} = \begin{bmatrix} 8 & 13 \\ 12 & 16 \end{bmatrix} \tag{2-12}$$

Definition 2-2.3. Matrix Exponentials
Max-plus Exponentials can be most easily defined recursively. For an exponential the matrix needs to be square so let $A \in \mathbb{R}_\varepsilon^{n \times n}$

$$A^{\otimes k} = A^{\otimes k-1} \otimes A \tag{2-13}$$

2-3 Notation Conventions

This section is still based off [2].

Definition 2-3.1. Zero Matrix
MPL algebra also has a notation convention for the a matrix filled with ε. In this case an $n \times n$ square zero matrix would be notated as $\mathcal{E}_n$ and an $m \times r$ sized version would be notated as $\mathcal{E}_{m \times r}$.

Definition 2-3.2. Alternative zero notation
In some literature e is written in stead of 0. In this thesis 0 will be used in the conext of conventional algebra and e in the context of MPL algebra.

$$e \stackrel{\text{def}}{=} 0 \tag{2-14}$$

Definition 2-3.3. Unit Matrix
The unit matrix is defined as a square $\mathcal{E}$ matrix with 0 on the diagonal. So an $n \times n$ unit matrix would be notated as E_n

Example 2-3.1. Unit Matrix
This means E_3 would work out as:

$$E_3 = \begin{bmatrix} 0 & \varepsilon & \varepsilon \\ \varepsilon & 0 & \varepsilon \\ \varepsilon & \varepsilon & 0 \end{bmatrix} \tag{2-15}$$

Definition 2-3.4. Unit Vector
The unit vector η is defined as a column vector with only 0 valued elements. So a height n unit vector would be:

$$\eta_n = \mathcal{E}_{n \times 1} \oplus 0 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2-16}$$

Definition 2-3.5. Vector Sum
In some literature the maximum element of a vector is notated as $||x(k)||_{\oplus}$. This will be done in this thesis as well.

$$||x(k)||_{\oplus} \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i(k) \tag{2-17}$$

2-4 State Space Models

MPL algebra is mainly used for describing Discrete Event Systems (DES). DES can be modeled using MPL algebra in a way that parallels a conventional state-space description. With the main difference being that DES don't deal in physical quantities like speed, temperature or acceleration but rather in time. So the state contains a time instance of arrival and the next calculation step is not some discrete time step, like say a second, but rather the next 'event' in the DES. Definition 2-4.1 through Definition 2-4.3 are based on [2] and everything after that can be found in [6].

Definition 2-4.1. Autonomous State Space Model
The simplest system is autonomous without an output. $A(k)$ might vary over events which is something that has been discussed in Section 2-7. Up to that chapter it can be assumed that $A(k)$ is constant so $A(k) = A \; \forall k$.

$$\begin{cases} x(k+1) &= A(k) \otimes x(k) \\ x(0) &= x_0 \end{cases} \tag{2-18}$$

With $A \in \mathbb{R}_{\varepsilon}^{n \times n}$.

Definition 2-4.2. Graph A matrix relationship
Any weighted directed graph has a corresponding A matrix. For a graph with vertices $1, 2, ..., n$ the corresponding matrix is size $A \in \mathbb{R}_{\varepsilon}^{n \times n}$. With a_{ij} equal to the edge-weight from vertex j to vertex i. Wherever there is no connection $a_{ij} = \varepsilon$.

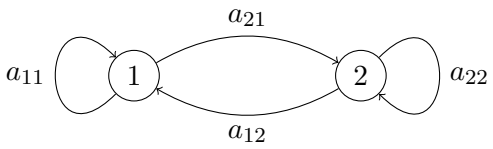

Figure 2-1: Graph corresponding to A: $\mathscr{G}(A)$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{2-19}$$

Example 2-4.1. A simple autonomous system
In order to illustrate how one might construct such a system lets look at a simple train line. Say we have two train stations, station 1 and 2 that both have a connection to a third station. From this station 3 a train departs to station 4, but this train can only depart after both trains from station 1 and 2 have arrived to provide a crossover connection. This system is shown in Figure 2-2 along the edges of the graph travel-time is notated in hours.

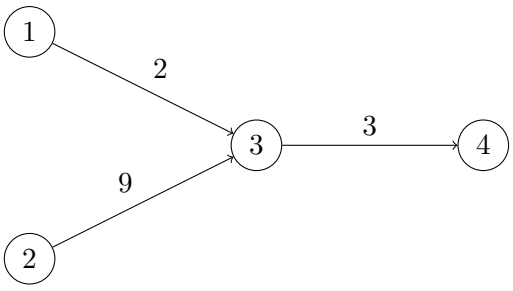

Figure 2-2: A simple system

It is possible to construct A from this graph using Definition 2-4.2. Doing so results in the A matrix shown in (2-20).

$$A = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 2 & 9 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 3 & \varepsilon \end{bmatrix} \tag{2-20}$$

In order to see how this system develops over time we need to set an initial state. Lets say there are only two trains in the initial state. One train in station 1 that leaves after 4 hours and one in station 2 that leaves immediately.

$$x(0) = \begin{bmatrix} 4 \\ 0 \\ \varepsilon \\ \varepsilon \end{bmatrix} \tag{2-21}$$

With such a simple system we can now reason that the first train will arrive at station 3 after $4 + 2 = 6$ and the second after $0 + 9 = 9$. Since the train for station 4 can only depart after both have arrived the next train will only arrive in station 4 after $\max(4+2, 0+9)+3 = 12$. We can do the same calculation using (2-18) to find $x(1)$ and $x(2)$.

$$x(1) = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 2 & 9 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 3 & \varepsilon \end{bmatrix} \otimes \begin{bmatrix} 4 \\ e \\ \varepsilon \\ \varepsilon \end{bmatrix} = \begin{bmatrix} \varepsilon \\ \varepsilon \\ 9 \\ \varepsilon \end{bmatrix}, x(2) = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 2 & 9 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 3 & \varepsilon \end{bmatrix} \otimes \begin{bmatrix} \varepsilon \\ \varepsilon \\ 9 \\ \varepsilon \end{bmatrix} = \begin{bmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \\ 12 \end{bmatrix} \tag{2-22}$$

As can be seen from the example the state vector $x(k)$ contains time instances at which processes begin. So in the case of $x(0)$ the first train could start driving 4 hours after t_0 whereas the other could leave immediately. And k does not signify time but rather counts events. There is no constant distance between events and in the case of $x(0)$ one event can contain two trains leaving 4 hours apart in time. But in the context of the DES both trains had to arrive at station 3 before the next possible event could happen, namely the connecting train leaving from station 3.

Definition 2-4.3. Input Dependent State Space Model
The system definition given in (2-18) can be extended with an input and output. Both $u(k)$ and $y(k)$ can be drawn in the graph as extra vertices. With $u(k)$ signifying time instances at which new input is given to the system, and $y(k)$ signifying time instances at which finished product leaves the system.

$$\begin{cases} x(k) &= A(k) \otimes x(k-1) \oplus B(k) \otimes u(k) \\ y(k) &= C(k) \otimes x(k) \\ x(0) &= x_0 \end{cases} \tag{2-23}$$

With $A \in \mathbb{R}_{\varepsilon}^{n \times n}, B \in \mathbb{R}_{\varepsilon}^{n \times m}, C \in \mathbb{R}_{\varepsilon}^{l \times n}$ where m is the number of edges connected to the vertex $u(k)$ and l is the number of edges connected to $y(k)$

Algorithm 2-4.1. Deriving a State Space Model for JIT systems with production times

1. Set n equal to the number of non-input -output vertices in the graph. Set m equal to the number of input vertices. Set l equal to the number of output vertices. Then number the input vertices as the set $U = \{u_1(k), u_2(k)..., u_m(k)\}$, and do the same for $Y = \{y_1(k), y_2(k), ..., y_l(k)\}$.
2. Create empty matrices $A \in \mathbb{R}_{\varepsilon}^{n \times n}, B \in \mathbb{R}_{\varepsilon}^{n \times m}, C \in \mathbb{R}_{\varepsilon}^{l \times n}$

3. Start a counter at $l = 1$

4. Find a vertex v_l that only has an incoming edge $e_{U,l}$ from input vertices U. Set $a_{ll} = p_i$ and $a_{l\bullet} = \varepsilon$ with $\bullet = \{1, 2, ..., n\} \triangle \{l\}$. For all $u_g(k) \in U$ set $b_{lg} = t_{g,l}$. Remember that when there is no directed edge $e_{i,j}$ travel-time is by definition $t_{i,j} = \varepsilon$. Increase counter $l = l + 1$ and repeat if there are more vertices that meet the first criterion.

5. Set $V = \{v_1, ..., v_{l-1}\}$. Find a vertex v_l that only has incoming edges from $U \cup V$. Then:

$$a_{ij} = \bigoplus_{m=1}^{l-1} p_m \otimes t_{m,l} \otimes a_{mj}, \quad b_{lg} = t_{g,l} \oplus \bigoplus_{m=1}^{l-1} p_m \otimes t_{m,l} \otimes b_{mg}$$

   Increase counter $l = l + 1$ repeat if $l \leq n$.

6. Set $c_{ij} = p_j \otimes t_{j,y_i}$

Example 2-4.2. Just In Time Production System
Throughout later sections in this chapter we will discuss system properties that can be derived using the state-space model. In the examples given going forward we will refer to the systems shown in Figure 2-3 and Figure 2-5. The first of which is a DES that does not contain any circuits. Often these types of DES describe Just In Time (JIT) production and as such they have an input, time instances at which raw materials are fed, and an output, time instances when finished product comes out. Say that we have a list of machines containing what they produce and require like Table 2-1. And something like a factory floor plan which would give an indication of travel time between machines. Then that information can be used to produce a graph like Figure 2-3. Generally travel time is notated next to edges and processing time next to vertices as can be seen in Figure 2-3.

Machine	**Input**	**Output**	**Processing Time**
1	Raw Mat.	Widget A	3 min
2	Raw Mat.	Widget B	5 min
3	Widget A, -B	Widget C	8 min
4	Widget C	Widget D	5 min
5	Widget C, -D	Final Prod.	1 min

Table 2-1: Production system machine dependencies

A point of note here is that processing- and transportation capacity are not explicitly considered in this model. In example, say the conveyor belt between machine 1 and 3 can transport only 10 widgets per hour but machine 1 produces 20 widgets per hour equaling the amount machine 3 needs to run at full capacity. In that case the output buffer of machine 1 will fill up, resulting in real world delays that are invisible to the MPL model.

Machine production capacity can be represented through processing time. In the case of this example machine 3 provides for both machine 4 and 5 if both were to need one widget C that would mean that machine 3 really produces 1 widget per 4 minutes but since it has to produce 2 total processing time is set to 8 minutes. This modeled holding back of production in some outputbuffer until enough is produced for the next event makes the model simple but has a downside. It is clear that if the first widget produced by machine 3 was sent to machine 4 and the second to machine 5 the total wait time from input to output would decrease by

4 minutes. This optimization of sending to machine 4 first, could be modeled by decreasing travel time from machine 3 to machine 4 by 4 minutes. However there is of yet no good way to model such an optimization if travel time between machines is shorter than the time gained by optimization.

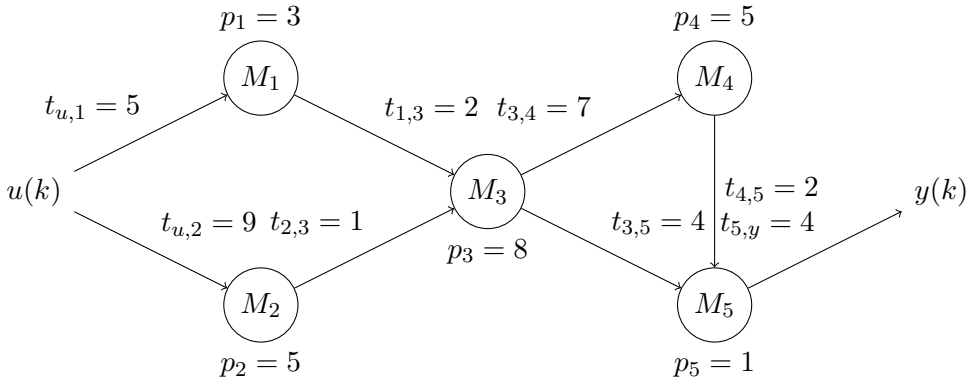

Figure 2-3: The graph corresponding to the production system

In this case it is possible to use Algorithm 2-4.1 to find the A, B and C matrix, however for the sake of the example we will work it out in detail. Before the first two machines can start operating they require raw materials $t_{u,i} \otimes u(k)$ and completion of any previous task $p_i \otimes x(k)$.

$$
\begin{aligned}
x_1(k) &= 3 \otimes x_1(k-1) \oplus 5 \otimes u(k) \\
x_2(k) &= 5 \otimes x_2(k-1) \oplus 9 \otimes u(k)
\end{aligned}
$$

Machines after that require completion of previous tasks $x_i(k-1) \otimes p_i$ and finished product of all previous machines $\bigoplus^J x_j(k) \oplus p_j \oplus t_{j,i}$ with J the set of all nodes with a directed edge pointing to vertex i.

$$
\begin{aligned}
x_3(k) &= 3 \otimes 2 \otimes x_1(k) \oplus 5 \otimes 1 \otimes x_2(k) \oplus 8 \otimes x_3(k-1) \\
&= 8 \otimes x_1(k-1) \oplus 11 \otimes x_2(k-1) \oplus 8 \otimes x_3(k-1) \oplus 15 \otimes u(k) \\
x_4(k) &= 8 \otimes 7 \otimes x_3(k) \oplus 5 \otimes x_4(k-1) \\
&= 23 \otimes x_1(k-1) \oplus 26 \otimes x_2(k-1) \oplus 23 \otimes x_3(k-1) \oplus 5 \otimes x_4(k-1) \oplus 5 \otimes u(k) \\
x_5(k) &= 8 \otimes 4 \otimes x_3(k) \oplus 5 \otimes 2 \otimes x_4(k) \oplus 1 \otimes x_5(k-1) \\
&= 30 \otimes x_1(k-1) \oplus 33 \otimes x_2(k-1) \oplus 30 \otimes x_3(k-1) \oplus 12 \otimes x_4(k-1) \oplus 1 \otimes x_5(k-1) \oplus 45 \otimes u(k) \\
y(k) &= 1 \oplus 4 \otimes x_5(k)
\end{aligned}
$$

This set of equations can then be written in state space form (2-24).

$$A_{ps} = \begin{bmatrix} 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 5 & \varepsilon & \varepsilon & \varepsilon \\ 2 & 11 & 8 & \varepsilon & \varepsilon \\ 8 & 26 & 23 & 5 & \varepsilon \\ 23 & 33 & 30 & 12 & 1 \end{bmatrix}, B_{ps} = \begin{bmatrix} 5 \\ 9 \\ 15 \\ 30 \\ 45 \end{bmatrix}, C_{ps} = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & 5 \end{bmatrix} \tag{2-24}$$

Definition 2-4.4. Schedule Dependent State Space Model
Another system that can be modeled using MPL algebra are schedule driven systems. These systems don't have an input $u(k)$ or output $y(k)$ in a physical sense. Rather their input is usually some kind of schedule $d(k)$. Logistic systems considered in this paper have static schedules. Static schedules repeat every event and thus are commonly defined $d(k) = d(0) \otimes T^{\otimes^k}$ with T the time period over which the schedule repeats. Logistic system output is arrival times at some set of vertices selected through C.

$$\begin{cases} x(k+1) &= A(k) \otimes x(k) \oplus d(k+1) \\ y(k) &= C(k) \otimes x(k) \\ x(0) &= x_0 \end{cases} \tag{2-25}$$

With $A \in \mathbb{R}_{\varepsilon}^{n+r \times n+r}, C \in \mathbb{R}_{\varepsilon}^{l \times n+r}$. To note here is the difference of $x(k)$ spanning $n+r$ instead of just n as is the case in Definition 2-4.3. In logistic systems n counts (logistical) connections, whereas r counts auxiliary variables, see Algorithm 2-4.3, that exist to make sure that the schedule is realistic, see Definition 2-6.3.

Definition 2-4.5. Transportation and Transfer Matrices
In a logistic system where transfer time q_i needs to be considered it is not possible to simply use Definition 2-4.2. One way is to just add the line specific transfer time to the line specific traveltime t_i for a total delay until the next event. Another way of doing the same calculation that allows for greater flexibility down the road is splitting up transfer time and travel time into two separate matrices $N \in \mathbb{R}_{\varepsilon}^{n \times n}$, $M \in \mathbb{R}_{\varepsilon}^{n \times n}$. N is defined as $n_{ii} = q_i$ with all other elements ε. M is defined with $m_{ji} = t_i$ for all vertices where some line j has to wait for line i before it can leave. All other elements of M, meaning those that do not correspond to any vertex on the graph, are set $m_{ji} = \varepsilon$.

$$N \otimes M = \begin{bmatrix} q_1 & \varepsilon & \dots & \varepsilon \\ \varepsilon & q_2 & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon & \varepsilon & \dots & q_n \end{bmatrix} \otimes \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ m_{21} & m_{22} & \dots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nn} \end{bmatrix} = A \tag{2-26}$$

Theorem 2-4.1. Order-p Model
For some systems, mostly those with travel-times in excess of T, not all trains manage to make it to their required station within one event. In reality this often means that some connection is so long that there are multiple trains travelling on one piece of track that depart in one event and arrive two or more events later. This can be done by splitting up the original M into some set M_1 through M_p with p the order of said system [6]. These M_1 through M_p can be found using Algorithm 2-4.2.

$$\begin{cases} x(k+1) &= \bigoplus_{l=1}^{p} N \otimes M_l \otimes x(k-l+1) \oplus d(k+1) \\ y(k) &= C(k) \otimes x(k) \\ x(0) &= x_0 \\ &\vdots \\ x(1-p) &= x_{1-p} \\ d(k) &= d(0) \otimes T^{\otimes^k} \end{cases} \tag{2-27}$$

It is possible for there to be gaps in the numbering from M_1 through M_p, nevertheless the order of the model is that of p which can thus be higher than the number of M_l matrices.

Algorithm 2-4.2. Finding M_1 through M_p
Start with the given matrices $M \in \mathbb{R}_{\varepsilon}^{n \times n}$ and $N \in \mathbb{R}_{\varepsilon}^{n \times n}$

1. Create a $\mathcal{E}$ matrix L of size $n \times n$.
2. For every element of $N \otimes M$ that is not ε calculate $d_i(0) - d_j(0) - (N \otimes M)_{ij}$ and then pick $l \in \mathbb{N}$ such that $-l \cdot T \leq d_i(0) - d_j(0) - (N \otimes M)_{ij} < (1-l) \cdot T$ is true. Set L_{ij} equal to the l found for every element of $(N \otimes M)_{ij}$.
3. For all unique values in L that are not ε create an $\mathcal{E}$ matrix $M_l \in \mathbb{R}_{\varepsilon}^{n \times n}$.
4. For every element of $L_{ij} \neq \varepsilon$ set $(M_{L_{ij}})_{ij} = M_{ij}$

This results in some set of matrices M_1 through M_p that can be used to put the system into the form described in Theorem 2-4.1. It is possible that one finds some $-l \cdot T \leq d_i(0) - d_j(0) - (N \otimes M)_{ij} < (1-l) \cdot T$ for which $l \leq 0$ in this case the system has failed the condition that no vehicle is to wait for more than T time units. And just as any value $l-1$ indicates the number of vehicles short, $-(l-1)$ indicates the number of vehicles in excess.

Algorithm 2-4.3. Finding Auxiliary Variables
In order to simplify an order-p system one step, meaning to go to an order-$(p-1)$ system we will need to expand M_{p-1} with auxiliary variables.

1. Define a set $N(l)$ containing the collumcounts for all columns of M_l with some element not equal to ε.

$$N(l) \stackrel{\textbf{def}}{=} \{j \in \{1, ..., n\} | \exists i \in \{1, ..., n\} \text{ such that } (M_l)_{ij} \neq \varepsilon\} \tag{2-28}$$

2. Calculate the number of auxiliary variables required r_l^*. Where we set $l = p-1$. Which means $|N(l)|$ is equal to the number of columns in M_l with some element not equal to ε.

$$r_l^* = (l-1) \cdot |N(l)| \tag{2-29}$$

3. Expand $d(k)$, N and M_1 through M_{p-1} as follows

$$\widehat{d(k)} = \begin{bmatrix} d(k) \\ \mathcal{E}_{1 \times r} \end{bmatrix} \widehat{N} = \begin{bmatrix} N & \mathcal{E}_{n \times r} \\ \mathcal{E}_{r \times n} & E_r \end{bmatrix}, \ \widehat{M_l} = \begin{bmatrix} M_l & \mathcal{E}_{n \times r} \\ \mathcal{E}_{r \times n} & \mathcal{E}_{r \times r} \end{bmatrix} \forall \ 1 \leq l \leq p-1 \tag{2-30}$$

with $r = r_l^*$

4. Start a counter $s = 1$

5. Set t to be equal to item s from the set $N(p)$. Set $\widehat{(M_{p-1})}_{t,n+s} = T - v$ and set $\widehat{(M_{p-1})}_{n+s,1\ldots n} = (M_p)_{s,1\ldots n} - (T - v)$ and lastly $\widehat{d(k)}_{n+s} = d(k)_t - v$. Here $v = 0$ can always be chosen, but this may result in negative elements in $\widehat{(M_{p-1})}$ so for human legibility it can be useful to shift $\widehat{(M_{p-1})}_{t,n+s}$ back in time some amount v. If $s < |N(p)|$ do $s = s + 1$ and repeat step 5.

6. with the system now decreased by one order set p to be equal to the order of the new highest order matrix $\widehat{(M_{p-1})}$. If $p > 1$ go to step 1.

Example 2-4.3. Simple Railway Network
Let us say we have a rail network that spans five cities, city A through D. This network connects all five cities together with seven train lines. The scheduling is hourly as such departure times are the same modulo $T = 60$ minutes. All of the rail network properties can be found in Table 2-2.

Train	From	To	Departure modulo 60	Travel Time t_i	Changeover Time q_i
1	A	B	27	45	2
2	B	C	15	36	2
3	B	D	20	42	2
4	C	A	25	79	1
5	D	C	17	50	2
6	D	E	06	47	2
7	E	C	55	24	3

Table 2-2: Train line travel time and dependencies

Figure 2-4 shows what a simplified map of Table 2-2 this system might look like. However it should not be confused with the graph for the MPL system.

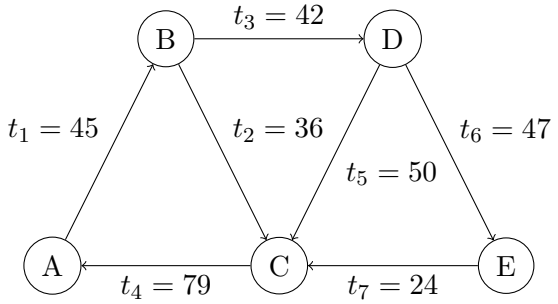

Figure 2-4: Simplified Map of Stations and Connections

Using Definition 2-4.5 and the information from Table 2-2 we can find M and N to calculate A as is done in (2-31)

$$N \otimes M = \begin{bmatrix} 2 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 2 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 2 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 3 \end{bmatrix} \otimes \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & 79 & \varepsilon & \varepsilon & \varepsilon \\ 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 36 & \varepsilon & \varepsilon & 50 & \varepsilon & 24 \\ \varepsilon & \varepsilon & 42 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 42 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 47 & \varepsilon \end{bmatrix} \tag{2-31}$$

$$= \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & 81 & \varepsilon & \varepsilon & \varepsilon \\ 47 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 47 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 37 & \varepsilon & \varepsilon & 51 & \varepsilon & 25 \\ \varepsilon & \varepsilon & 44 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 44 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 50 & \varepsilon \end{bmatrix} = A^\star \tag{2-32}$$

The actual graph for the system has the train connections as vertices not the stations. And crossover time added to travel time between before the next train in the circuit can leave is written next to the edges. This graph can be seen in Figure 2-5.

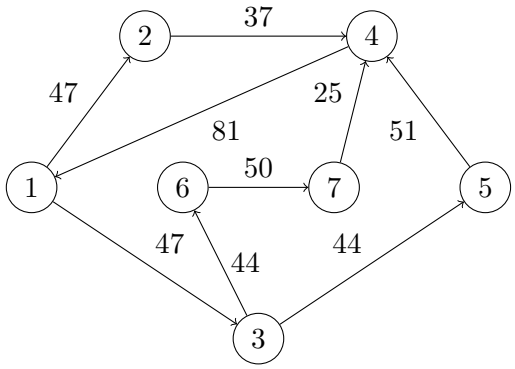

Figure 2-5: Actual System Graph

But we do not yet know if this system is realistic. In order to test this, and do many other types of analysis discussed in Chapter **??**, we have to make sure the system is of p-order 1. This is done by applying Algorithm 2-4.2 to N and M. If we find that all elements of L are equal to ε or 1 the system was already of P order 1.

$$\begin{array}{lccccccl} & -l \cdot T & \leq & d_i(0) - d_j(0) - (N \otimes M)_{ij} & < & (1-l) \cdot T & & \\ A^\star_{14} = 81, & -120 & \leq & 12 - 45 - 81 = -114 & < & -60, & L_{14} = 2 & \\ A^\star_{21} = 47, & -60 & \leq & 11 - 12 - 47 = -57 & < & 0, & L_{21} = 1 & \\ & & & \vdots & & & & \\ A^\star_{76} = 50, & -60 & \leq & 55 - 6 - 50 = -1 & < & 0, & L_{50} = 1 & \end{array} \tag{2-33}$$

With the first part of the algorithm done we can already see that the system is not of P order 1 yet. We find two M_l matrices are required to adequately describe the system:

$$M_1 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 36 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 24 \\ \varepsilon & \varepsilon & 42 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 42 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 47 & \varepsilon \end{bmatrix}, \; M_2 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & 79 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 50 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix} \tag{2-34}$$

Using Algorithm 2-4.3 we can now find the required auxiliary variables to simplify M_1 and M_2 into some $\widehat{M}$. We will only need to loop through the algorithm once since we only require one simplification step.

There are two columns in M_2 that contain a non ε element namely $(M_2)_{14}$ and $(M_2)_{54}$ so $|N(l)| = 2$. We can use (2-29) to calculate that 2 dummy trains are required.

$$r_l^* = (l-1) \cdot |N(l)| = (2-1) \cdot 2 = 2 \tag{2-35}$$

Now that we know how many dummy trains are required we can implement them in a new $\widehat{M}$. The goal with these dummy trains is to allow for 2 different trains of the same line to be on one rail track at the same time. Since for example the track from Station C to Station A is 79 minutes of driving it is not possible for one train to make it within one schedule cycle $T = 60$ minutes. This should be fine as there will just be 2 trains on the track for $79-60 = 19$ minutes as the second train departs from station C before the first one arrives at station A. However we can't model this in a p order 1 model without some alterations. The trick that will solve this issue is to add a non-existent station somewhere along the track between station A and station C. With this dummy station placed somewhere between minute 19 and minute 60 it is possible to simulate two trains on the station A to station C track. With the dummy station included we model the one longer track as two shorter tracks which never have two trains on them at the same time.

$$\begin{aligned} \widehat{d_8(k)} &= d_1(k) = 12 \otimes 60^{\otimes^k} \\ \widehat{M}_{48} &= T = 60 \\ \widehat{M}_{81} &= (M_2)_{14} - T = 79 - 60 = 19 \end{aligned} \tag{2-36}$$

The first auxilliary variable (2-36) could be introduced without trouble. However the second does require a shift of 15 minutes in order to prevent negative elements in $\widehat{M}$. The model would work fine leaving them in, but it is counterintuitive for a train to travel back in time over the course of an event step. Thus it is better to shift the dummy train back down the line by 15 minutes so it can drive for 5 minutes before arriving at its destination rather than driving back in time 10 minutes.

$$\begin{aligned} \widehat{d_9(k)} &= d_4(k) - 15 = 30 \otimes 60^{\otimes^k} \\ \widehat{M}_{59} &= T - 15 = 45 \\ \widehat{M}_{94} &= (M_2)_{14} - (T-15) = 50 - 45 = 5 \end{aligned} \tag{2-37}$$

Completing $\widehat{M}, \widehat{d(k)}$ with the dummy variables and expanding $\widehat{N}$ with 0 minute transition times for the dummy stations, gets us the required matrices for a P-order model of order 1.

$$\widehat{N} = \begin{bmatrix} N & \mathcal{E}_{n\times r} \\ \mathcal{E}_{r\times n} & E_r \end{bmatrix}, \widehat{M} = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 19 & \varepsilon \\ 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 36 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 24 & \varepsilon & 5 \\ \varepsilon & \varepsilon & 42 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 42 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 47 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 60 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}, \widehat{d(k)} = \begin{bmatrix} 27 \\ 15 \\ 20 \\ 25 \\ 17 \\ 6 \\ 55 \\ 27 \\ 10 \end{bmatrix} \otimes 30^{\otimes^k} \tag{2-38}$$

We can then Simplify $\widehat{N} \otimes \widehat{M} = A_{ls}$ and define a B_{ls} and C_{ls} to put the system into regular MPL state-space form found in Definition 2-4.4.

$$A_{ls} = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 21 & \varepsilon \\ 47 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 47 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 37 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 25 & \varepsilon & 6 \\ \varepsilon & \varepsilon & 44 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 44 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 50 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 60 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}, d_{ls}(k) = \begin{bmatrix} 27 \\ 15 \\ 20 \\ 25 \\ 17 \\ 6 \\ 55 \\ 27 \\ 10 \end{bmatrix} \otimes 30^{\otimes^k} \tag{2-39}$$

$$C_{ls} = \begin{bmatrix} E_n & \mathcal{E}_{n\times r} \end{bmatrix} \text{ with } n = 7 \text{ and } r = 2$$

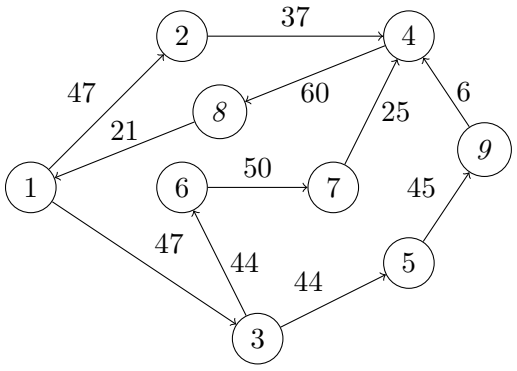

Figure 2-6: System Graph with auxiliary variables *8* and *9* included

2-5 Basic System Properties

Again everything discussed in this section can also be found in [6].

Definition 2-5.1. Paths and Circuits
A path is a route through some directed graph which is denoted by naming the vertices passed: $\rho = v_1 \to v_2 \to ... \to v_l$. A circuit is a path that contains no vertex twice and loops back on itself: $c = v_1 \to v_2 \to ... \to v_l \to v_1$

Definition 2-5.2. Path and Circuit properties
Paths and circuits have certain properties that can be used for analysis. First is weight $|\rho|_w$ which is defined as the sum of edge- and vertex-weights, including the starting vertex and excluding the final vertex, along a path or circuit. Secondly, there is length $|\rho|_l$ which is defined as the number of edges in any path. Using these two a mean weight can be calculated $\frac{|\rho|_w}{|\rho|_l} = |\rho|_m$.

For circuits weight $|c|_w$ is calculated using all edge- and vertex-weights included in the circuit as there are no start- or final vertices. Similarly for circuit length $|c|_l$ which is found by counting all edges included in the circuit. Both circuit weight and circuit length can then be used to calculate circuit mean $|c|_m$ in the same way as described for paths.

Definition 2-5.3. Irreducibility
If in $\mathscr{G}(A)$ a path exists between any two vertices that graph is called strongly connected and its corresponding matrix A is irreducible. Alternatively using conventional algebra, if there exists no transformation matrix P such that P^TAP is an upper-triangular matrix, A is irreducible as well.

Example 2-5.1. Irreducibility
Figure 2-3 is clearly not strongly connected as there is no way to reach any other vertices from v_5. Which we can confirm using conventional algebra:

$$P^TA_{nl}P = \begin{bmatrix}0&0&0&0&1\\0&0&0&1&0\\0&0&1&0&0\\0&1&0&0&0\\1&0&0&0&0\end{bmatrix}\begin{bmatrix}3&\varepsilon&\varepsilon&\varepsilon&\varepsilon\\\varepsilon&5&\varepsilon&\varepsilon&\varepsilon\\2&11&8&\varepsilon&\varepsilon\\8&26&23&5&\varepsilon\\23&33&30&12&1\end{bmatrix}\begin{bmatrix}0&0&0&0&1\\0&0&0&1&0\\0&0&1&0&0\\0&1&0&0&0\\1&0&0&0&0\end{bmatrix} = \begin{bmatrix}3&\varepsilon&2&8&23\\\varepsilon&5&11&26&33\\\varepsilon&\varepsilon&8&23&30\\\varepsilon&\varepsilon&\varepsilon&5&12\\\varepsilon&\varepsilon&\varepsilon&\varepsilon&1\end{bmatrix} \tag{2-40}$$

Figure 2-6 however, is irreducible as there is a path from every vertex to every vertex. This is easily shown by looking at the three intersecting circuits that loop through v_1, v_4 and v_8.

$$\begin{aligned} c_1 &= v_1 \to v_2 \to v_4 \to v_8 \to v_1 \to ... \\ c_2 &= v_1 \to v_3 \to v_5 \to v_9 \to v_4 \to v_8 \to v_1 \to ... \\ c_3 &= v_1 \to v_3 \to v_6 \to v_7 \to v_4 \to v_8 \to v_1 \to ... \end{aligned} \tag{2-41}$$

Definition 2-5.4. Critical Circuit
The circuit with the maximum circuit mean contained within the graph $\mathscr{G}(A)$ of some sytem is to be denoted as the critical circuit $c_\star$ of the system. There can be multiple critical circuits if more than one circuit has the same circuit mean equal to the maximum circuit mean. This value is also referred to as maximal circuit mean like in [7].

Definition 2-5.5. Eigenvalues
If for a square matrix A there is some vector $v \neq \varepsilon$ and scalar value λ for which $A \otimes v = \lambda \otimes v$ then λ is an eigenvalue for A and v an eigenvector.

Theorem 2-5.1. Irreducibility and Eigenvalues
If the A matrix is irreducible it has a unique eigenvalue. This eigenvalue is equal to the mean of the critical circuit of the system.

Algorithm 2-5.1. Eigenvalue Power Algorithm
This method [8] for finding eigenvalues λ and corresponding eigenvectors v can only be applied to irreducible A matrices.

1. Take an arbitrary vector $x_0 \neq \mathcal{E}_{n\times 1}$, in most cases $x_0 = \eta_n$ is easiest.
2. start counter i = 1
3. Find $x(i)$ by calculating $x(i) = A \otimes x(i-1)$
4. start counter $j = i - 1$
5. Test if $x(i) - x(j) = \eta_n \otimes \lambda^{\otimes^{i-j}}$. If it is possible to express the difference between the two x vectors this way go to step 8.
6. If $j > 0$ do $j = j - 1$ and go to step 5.
7. Increase counter $i = i + 1$ and go to step 3.
8. The unique eigenvalue of the system is the λ found with $j = k_0$. These values can then be used to calculate $A^{\otimes^{k+c}} = \lambda^{\otimes^c} \otimes A^{\otimes^k}$ as discussed in Theorem 2-5.1.

There are more algorithms for finding eigenvalues like the policy iteration algorithm [7], or ones optimized for special eigenvalue problems [9].

Example 2-5.2. Eigenvalue Power Algorithm and Critical Circuit
We can't apply the eigenvalue power algorithm to the JIT production example system as it is not irreducible. However since the railway network example is irreducible we can calculate $x(1)$ through $x(12)$ using (2-39) with $x_0 = \eta_9$.

$x(0)$	$x(1)$	$x(2)$	$x(3)$	$x(4)$	$x(5)$	$x(6)$	$x(7)$	$x(8)$	$x(9)$	$x(10)$	$x(11)$	$x(12)$
0	21	81	118	165	200	247	283	330	365	412	448	495
0	47	68	128	165	212	247	294	330	377	412	459	495
0	47	68	128	165	212	247	294	330	377	412	459	495
0	37	84	119	166	202	249	284	331	367	414	449	496
0	44	91	112	172	209	256	291	338	374	421	456	503
0	44	91	112	172	209	256	291	338	374	421	456	503
0	50	94	141	162	222	259	306	341	388	424	471	506
0	60	97	144	179	226	262	309	344	391	427	474	509
0	45	89	136	157	217	254	301	336	383	419	466	501

We can find the first case for which $x(i) - x(j) = \eta_n \otimes \lambda^{\otimes^{i-j}}$ is true at step $i = 10$ for the

sake of showing the periodicity we show up to $x(14)$.

$x(9)-x(6)$	$x(10)-x(6)$	$x(11)-x(6)$	$x(12)-x(6)$	$x(13)-x(6)$	$x(14)-x(6)$
118	165	201	248	283	330
130	165	212	248	295	330
130	165	212	248	295	330
118	165	200	247	283	330
118	165	200	247	283	330
118	165	200	247	283	330
129	165	212	247	294	330
129	165	212	247	294	330
129	165	212	247	294	330

Working out $x(10)-x(6)=\eta_9\otimes\lambda^{\otimes^{10-6}}$ gets us $\lambda=\frac{165}{4}=41.25$ Looking at the 3 circuits contained in $\mathscr{G}(A_{ls})$ defined in (2-41) we find:

Circuit	Weight ($\lvert c\rvert_w$)	Length ($\lvert c\rvert_l$)	Mean Weight ($\lvert c\rvert_m$)
c_1	165	4	$41\frac{1}{4}$
c_2	223	6	$37\frac{1}{6}$
c_3	247	6	$41\frac{1}{6}$

With $c_1=c_\star$ as $|c_1|_m=\lambda$ and as expected the first j that worked was $j=6=k_0$ which makes intuitive sense looking at $\max(|c_1|_l,|c_2|_l,|c_3|_l)=6$ and as expected $i-j=4=|c_\star|_l$

Theorem 2-5.2. System Periodicity
The eigenvalue of an irreducible A matrix can be used for calculating MPL exponentials. If $A\in\mathbb{R}_\varepsilon^{n\times n},\ k_0\in\mathbb{N},\ c\in\mathbb{N}\cup\{0\}$ then for $\forall k\geq k_0$

$$A^{\otimes^{k+c}}=\lambda^{\otimes^c}\otimes A^{\otimes^k} \tag{2-42}$$

Where the periodicity is equal to the length of the critical circuit $c=|c_\star|_l$. And the minimum number of events before the system can be used is equal to the longest circuit of the system: $k_0=\bigoplus_i^{\mathcal{C}}|c_i|_l$ witch $\mathcal{C}$ a set containing all unique circuits of the system.

In systems with multiple critical circuits periodicity depends on how and if they intersect. Two intersecting periodic circuits can have a lower periodicity than the length of either one. In any case periodicity is always the longest critical circuit.

2-6 Max-Plus Linear Scheduling

With the basics out of the way we can now look at MPL system properties that are useful for scheduling. Everything discussed in this section can be found in [6].

Definition 2-6.1. Minimum Production Time
The minimum production time is how long it takes to go from input to output assuming empty buffers. Assuming empty buffers means $x_0=\varepsilon$ which means that $y(1)=C\otimes x(1)$ simplifies to $y(1)=C\otimes B\otimes u(1)$. In this case for $u(1)=0$, $y(1)$ is equal to the production time starting with empty buffers.

Definition 2-6.2. Bottleneck
In imperfectly balanced production systems one or more machines are the bottleneck [10]. This means that the overall performance of the system can only be improved by improving the performance of these machines.

Algorithm 2-6.1. Finding Bottlenecks
In schedule driven systems the bottleneck is the critical circuit. But in systems without any circuits we can find the bottleneck by looking for where synchronization occurs. Imagine one were to feed a factory line like the JIT production system like Example 2-4.2 an infinite amount of raw material and perform no synchronization. You would see half finished product heaping up in one or more buffers with empty buffers in front of the machines after the bottleneck. With an MPL model this can be simulated by performing two event steps and seeing which machines are constantly occupied with work. Those machines are where the synchronization takes place since all machines in front of it have to wait for its buffer to empty out before starting another round of processing.

Definition 2-6.3. Realistic Schedule
For a schedule to be realistic it has to be possible to arrive on time for the next schedule when leaving in time relative to the current schedule. A system that conforms to this requirement is sometimes called a realistic system.

$$A \otimes d(k) \leq d(k+1), \quad \forall k \geq 0 \tag{2-43}$$

Example 2-6.1. Realistic Schedule
When we apply Definition 2-6.3 to our railway network example (2-39) we can calculate:

$$A_{ls} \otimes d_{ls}(0) = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 21 & \varepsilon \\ 47 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 47 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 37 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 25 & \varepsilon & 6 \\ \varepsilon & \varepsilon & 44 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 44 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 50 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 60 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 45 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix} \otimes \begin{bmatrix} 27 \\ 15 \\ 20 \\ 25 \\ 17 \\ 6 \\ 55 \\ 27 \\ 10 \end{bmatrix} = \begin{bmatrix} 48 \\ 74 \\ 74 \\ 80 \\ 64 \\ 64 \\ 56 \\ 85 \\ 62 \end{bmatrix} \leq \begin{bmatrix} 87 \\ 75 \\ 80 \\ 85 \\ 77 \\ 66 \\ 115 \\ 87 \\ 70 \end{bmatrix} = d_{ls}(1) \tag{2-44}$$

We find the schedule to be realistic when applied to this system.

Definition 2-6.4. Delay State Space
Since we know actual departure times $x(k)$ and the scheduled departure times $d(k)$ we can calculate the system delay as follows:

$$z(k) \stackrel{\text{def}}{=} x(k) - d(k) \tag{2-45}$$

Which evolves along:

$$\begin{aligned} z(k+1) &= x(k+1) - d(k+1) \\ &= (A \otimes x(k) \oplus d(k+1)) - d(k+1) \\ &= A \otimes x(k) - d(k+1) \oplus d(k+1) - d(k+1) \\ &= A \otimes x(k) - d(k+1) \oplus \eta_{n+r} \\ &\text{substituting } x(k) = z(k) + d(k) \\ &= A \otimes (z(k) + d(k)) - d(k+1) \oplus \eta_{n+r} \\ &= A \otimes (z(k) + d(0)) - d(1) \oplus \eta_{n+r} \\ &\text{substituting } d(k+1) = d(k) \otimes T \\ &= A \otimes (z(k) + d(0)) - d(0) \otimes T \oplus \eta_{n+r} \end{aligned}$$

Which means we can formulate a delay state-space model:

$$\begin{cases} z(k+1) &= A \otimes (z(k) + d(0)) - d(0) \otimes T \oplus \eta_{n+r} \\ \widetilde{y}(k) &= C \otimes z(k) \\ z(0) &= x_0 - d(0) \end{cases} \tag{2-46}$$

The largest delay in the system can then be written as.

$$\|z(k)\|_{\oplus} \tag{2-47}$$

Definition 2-6.5. Settling Time
The settling time of a system is the number of event steps k_s until $\|\widetilde{y}(k)\|_{\oplus} = 0 \; \forall k \geq k_s$. It is important not to depend on $\|\widetilde{y}(k)\|$ to check if the system has settled if $r > 0$ since the auxiliary component of $x(k)$ can still contain them. However if you only have access to $\widetilde{y}(k)$ and you know the p-order of the original system, it is still possible to calculate the settling time within a p sized margin. In this case going from the first case where $\widetilde{y}(k), \widetilde{y}(k+1), ..., \widetilde{y}(k+p-1)$ are all equal to zero the system settled somewhere between k and $k+p-1$.

Example 2-6.2. Settling Time
We can simulate an initial delay of 25 minutes on train 3 by setting:

$$x_0 = d(0) + z(0) = \begin{bmatrix} 27 \\ 15 \\ 20 \\ 25 \\ 17 \\ 6 \\ 55 \\ 27 \\ 10 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 25 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 27 \\ 15 \\ 45 \\ 25 \\ 17 \\ 6 \\ 55 \\ 27 \\ 10 \end{bmatrix}$$

Then calculating $x(1)$ through $x(3)$:

$$x(1) = \begin{bmatrix} 87 \\ 75 \\ 80 \\ 85 \\ 89 \\ 89 \\ 115 \\ 87 \\ 70 \end{bmatrix}, \; z(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 12 \\ 23 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; x(2) = \begin{bmatrix} 147 \\ 135 \\ 140 \\ 145 \\ 137 \\ 126 \\ 175 \\ 147 \\ 134 \end{bmatrix}, \; z(2) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}, \; x(3) = \begin{bmatrix} 207 \\ 195 \\ 200 \\ 205 \\ 197 \\ 186 \\ 235 \\ 207 \\ 190 \end{bmatrix}, \; z(3) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This results in $\|\widetilde{y}(0)\|_{\oplus} = 25, \; \|\widetilde{y}(1)\|_{\oplus} = 23, \; \|\widetilde{y}(2)\|_{\oplus} = 0, \; \|\widetilde{y}(3)\|_{\oplus} = 0$ this means $k_s = 2$. In this case it might seem like there is no need to look at $z(k)$ since the delay that shows up in the auxiliary variables does not later become visible in $\widetilde{y}(k)$. Had the initial delay on train three been 70 minutes then $\|\widetilde{y}(2)\|_{\oplus} = 0$ would still be the case, but since $z_9(2) = 49$ the delay propagates to $z_4(3) = 4$ and even $z_8(4) = 2$ until finally $\|z(5)\|_{\oplus} = 0$. However $z_8(k)$ is auxiliary and as such $\|\widetilde{y}(4)\|_{\oplus} = 0$ putting $k_s = 4$.

Theorem 2-6.1. Meta-Stability
Equation (2-43) can be written as:

$$\bigoplus_{j=1}^{n+r} A_{ij} d_j(k) \leq d_i(k+1) \tag{2-48}$$

Which rewrites to:

$$A_{ij} \leq d_i(k+1) - d_j(k) \tag{2-49}$$

If we take the maximum delay from the delay state space model (2-46). We can use the previous relation to substitute A_{ij}.

$$\begin{aligned}
\|z(k+1)\|_{\oplus} &= \|x(k+1) - d(k+1)\|_{\oplus} \\
&= \|(A \otimes x(k) \oplus d(k+1)) - d(k+1)\|_{\oplus} \\
&= \|A \otimes x(k) - d(k+1)\|_{\oplus} \oplus 0 \\
&= 0 \oplus \bigoplus_{i=1}^{n+r} \left(\bigoplus_{j=1}^{n+r} A_{ij} + x_j(k) \right) - d_i(k+1) \\
&\text{Here substitute } A_{ij} \\
&\leq 0 \oplus \bigoplus_{i=1}^{n+r} \left(\bigoplus_{j=1}^{n+r} d_i(k+1) - d_j(k) + x_j(k) \right) - d_i(k+1) \\
&= 0 \oplus \bigoplus_{i=1}^{n+r} \left(\bigoplus_{j=1}^{n+r} x_j(k) - d_j(k) \right) + d_i(k+1) - d_i(k+1) \\
&= 0 \oplus \bigoplus_{i=1}^{n+r} \left(\bigoplus_{j=1}^{n+r} x_j(k) - d_j(k) \right) \\
&= \|z(k)\|_{\oplus}
\end{aligned} \tag{2-50}$$

Thus a realistic schedule guarantees meta-stability. If there is no delay in the system it will not increase. And if the system is perturbed such that some delay is introduced, the maximum delay in the system will not increase.

Theorem 2-6.2. Stability
A schedule driven system is stable if all delays can be resolved within a finite number of events. This requires:

$$\begin{cases} \|z(k+h)\| & < & \|z(k)\| & \forall \|z(k)\| > 0,\ h < \infty \\ \|z(k+1)\| & = & 0 & \forall \|z(k)\| = 0 \end{cases} \tag{2-51}$$

The second is guaranteed for systems with a realistic schedule as is shown in Theorem 2-6.1 this same theory also at least guarantees that $\|z(k+1)\|_\oplus \leq \|z(k)\|_\oplus$. But to guarantee that

$$T - \lambda > 0 \tag{2-52}$$

Also assume we have an irreducible system with periodicity c and we use some $k \geq k_0$. Which means we can use (2-42). If we then substitute for λ we find:

$$\begin{aligned} A^{\otimes^{k+c}} &= \lambda^{\otimes^c} \otimes A^{\otimes^k} \\ &< T^{\otimes^c} \otimes A^{\otimes^k} \end{aligned} \tag{2-53}$$

$$\begin{aligned} \|z(k+c)\|_\oplus &= \|x(k+c) - d(k+c)\|_\oplus \\ &= \left\| \left(A^{\otimes^{k+c}} \otimes x_0 \oplus d(k+c) \right) - d(k+c) \right\|_\oplus \\ &= \left\| \left(\lambda^{\otimes^c} \otimes A^{\otimes^k} \otimes x_0 - d(k+c) \right) \oplus \left(d(k+c) - d(k+c) \right) \right\|_\oplus \\ &= \left\| \left(\lambda^{\otimes^c} \otimes A^{\otimes^k} \otimes x_0 - d(k+c) \right) \oplus 0 \right\|_\oplus \\ &\quad \text{By definition } \|z(k)\|_\oplus \geq 0 \text{ so we can write:} \\ &< \left\| T^{\otimes^c} \otimes A^{\otimes^k} \otimes x_0 - T^{\otimes^c} \otimes d(k) \right\|_\oplus \\ &= \left\| \left(T^{\otimes^c} - T^{\otimes^c} \right) \otimes A^{\otimes^k} \otimes x_0 - d(k) \right\|_\oplus \\ &= \|x(k) - d(k)\|_\oplus \\ &= \|z(k)\|_\oplus \end{aligned} \tag{2-54}$$

With a realistic schedule guaranteeing the maximum delay doesn't increase, this guarantees that over one system period the delay will decrease. Thus some finite delay will disappear from the system within a finite number of events for irreducible systems with a finite periodicity. All systems without a period have a trivial solution since $A^{\otimes^n} = 0$.

2-7 Switching Max-Plus Linear Systems

Up to this point we have only considered systems with static state space matrices. More complex systems can be described by having multiple state space matrices and actively switching between them [11]. For example if you have a production system with machines that can perform two types of operations you might have two A matrices: one for the multipurpose machines switched to operation type 1 and one for operation type 2.

Definition 2-7.1. Operation Mode
Systems can be in some set of operation modes $l(k) \in \{1, ..., n_L\}$. Matrices of in different modes are generally denoted: $A^{(l(k))}$.

Definition 2-7.2. Switching Input Dependent State-Space Model
We can rewrite the input dependent state-space model (2-23) to be switching:

$$\begin{cases} x(k) &= A^{(l(k))}(k) \otimes x(k-1) \oplus B^{(l(k))}(k) \otimes u(k) \\ y(k) &= C^{(l(k))}(k) \otimes x(k) \\ x(0) &= x_0 \end{cases} \tag{2-55}$$

With $A^{(l(k))} \in \mathbb{R}_\varepsilon^{n \times n}, B^{(l(k))} \in \mathbb{R}_\varepsilon^{n \times m}, C^{(l(k))} \in \mathbb{R}_\varepsilon^{l \times n}$.

Definition 2-7.3. Switching Schedule Dependent State-Space Model
We can rewrite the schedule system state-space model (2-25) to be switching:

$$\begin{cases} x(k) &= A^{(l(k))}(k) \otimes x(k) \oplus d(k)^{(l(k))} \\ y(k) &= C^{(l(k))}(k) \otimes x(k) \\ x(0) &= x_0 \end{cases} \tag{2-56}$$

With $A^{(l(k))} \in \mathbb{R}_\varepsilon^{n+r \times n+r}, C^{(l(k))} \in \mathbb{R}_\varepsilon^{l \times n+r}$.

Definition 2-7.4. Switching Variable
A system may switch operation mode for various reasons. Operation mode $l(k)$ can be found by calculating the switching variable $z(k)$. This switching variable may be a function of some previous state, the previous mode the input or some other control input $v(k)$:

$$z(k) = \Phi(x(k-1), l(k-1), u(k), v(k)) \in \mathbb{R}_\varepsilon^{n_z} \tag{2-57}$$

In the case of stochastic switching systems $v(k)$ can be a stochastic variable. If you then partition the scope of the Φ function $\mathbb{R}_\varepsilon^{n_z}$ into n_m not overlapping subsets $\mathscr{L}^{(i)}$ with $i = \{1, ..., n_m\}$. Then you can say that when $z(k) \in \mathscr{L}^{(i)}$ that $l(k) = i$.

Definition 2-7.5. Alternative Switching Variable Notation
There is another notation for calculating the switching variable more commonly used for stochastic systems [12]. Here we set $L(k) = \{1, 2, ..., n_L\}$ to be the set containing all possible operation modes. This notation also allows for the possibility to consider external conditions like $l(k-1), x(k-1), u(k), v(k)$.We can then note the odds that $l(k)$ will be equal to one specific operation mode in the set as:

$$P[l(k) = L(k) | l(k-1), x(k-1), u(k), v(k)] \tag{2-58}$$

Since this describes the odds that the switching variable is in a particular mode it is subject to:

$$\begin{aligned} &0 \leq P[l(k) = L(k) | l(k-1), x(k-1), u(k), v(k)] \leq 1 \\ &\sum_{l(k)=1}^{n_L} P[l(k) = L(k) | l(k-1), x(k-1), u(k), v(k)] = 1 \end{aligned} \tag{2-59}$$

To describe fully deterministic switching using this style of notation another condition is required:

$$P[l(k) = L(k) | l(k-1), x(k-1), u(k), v(k)] \in \{0, 1\} \tag{2-60}$$

Example 2-7.1. Extending the railway network example to be switching
Let us say that the original description of the rail network was incomplete. For some 10% of the rides from station D to station E or $(A_{ls}^{(1)})_{76} = 50$ instead of it taking 3 minutes crossover and a 47 minute drive the drive suddenly takes 78 minutes as a railway bridge is open forcing the train to wait. This means that every event step there is a 0.1 chance that the system operation mode is set $l(k) = 2$ with $(A_{ls}^{(2)})_{76} = 3 + 78 = 81$ and all other elements being the same $(A_{ls}^{(1)})_{ij} = (A_{ls}^{(2)})_{ij} \quad \forall i \neq 7 \wedge j \neq 6$.

When we apply the eigenvalue power algorithm (Algorithm 2-5.1) to $Als^{(2)}$ we find that the system has a different eigenvalue in the second operation mode.

$x(7)-x(6)$	$x(8)-x(6)$	$x(9)-x(6)$	$x(10)-x(6)$	$x(11)-x(6)$	$x(12)-x(6)$
21	81	118	187	231	278
47	68	128	165	234	278
47	68	128	165	234	278
37	106	150	197	218	278
44	91	112	172	209	278
44	91	112	172	209	278
69	113	160	181	241	278
60	97	166	210	257	278
69	113	160	181	241	278

Working out $x(12) - x(6) = \mathcal{E}_{n\times 1} \otimes \lambda^{\otimes^{12-6}}$ gets us $\lambda = \frac{278}{6} = 46\frac{1}{3}$. If we look at the 3 circuits contained in $\mathscr{G}(A_{ls}^{(2)})$ we find:

Circuit	Weight ($\lvert c\rvert_w$)	Length ($\lvert c\rvert_l$)	Mean Weight ($\lvert c\rvert_m$)
c_1	165	4	$41\frac{1}{4}$
c_2	223	6	$37\frac{1}{6}$
c_3	247	6	$46\frac{1}{3}$

Introducing the extra travel-time in $(A_{ls}^{(2)})_{76}$ changed the circuitweight of c_3 to such a degree that it is the critical circuit for $Als^{(2)}$ which also changes the periodicity of the system from 4 to 6.

2-8 Expected processing time for stochastic SMPL systems

In this chapter some new notation and predefined matrices are necessary for compactness and readability. These are the following:

$$\overline{B} = \max_{i,j} [B]_{ij}$$
$$[E]_{ij} = 0 \; \forall i, j$$

Using the definition of a Switching Max-Plus-Linear (SMPL) system (2-55) it is possible to calculate upcoming states in larger intervals than $k + 1$ through substitution:

$$\begin{aligned} x(k) &= A^{(\ell(k))} \otimes x(k-1) \\ &= A^{(\ell(k))} \otimes \left(A^{(l(k-1))} \otimes x(k-2) \right) \\ &= \left(\bigotimes_{m=1}^{N} A^{\ell(k+1-m)} \right) \otimes x(k-N) \end{aligned}$$

If we know all upcoming modes ℓ_m of the system in order we can write this max-plus matrix product more compactly on its own as:

$$\Gamma(N) = \bigotimes_{m=1}^{N} A^{(\ell_m)} \tag{2-61}$$

The upperbound $\overline{\Gamma(N)}$ of $\Gamma(N)$ will give a useful indication of $y(k)$ in most SMPL systems including the hypothetical sorting system. However since the systems we will consider are stochastic switching we do not know the exact mode for every step ℓ_m, only the odds distribution $\{p_1, ..., p_{\ell_m}, ..., p_L\}$. As such we are more interested in the expectation $\mathbb{E}\left[\overline{\Gamma(N)}\right]$, which can only be calculated by taking the odds weighted mean over the set of all permutations $\mathscr{L} \in \{\ell_1, ..., \ell_m, ..., \ell_N\}$:

$$\mathbb{E}\left[\overline{\Gamma(N)}\right] = \sum^{\mathscr{L}q} \left(\prod_{m=1}^{N} p_{\ell_m} \right) \cdot \left(\overline{\bigotimes_{m=1}^{N} A^{(\ell_m)}} \right) \tag{2-62}$$

Chapter 3

S-Matrix Approach

First we have to define the arbitrary matrices:

$$\begin{aligned} &S = \text{diag}_{\oplus}(s_1, ..., s_n) \quad \text{s.t.} \quad S_{ii} = s_i \; \forall i, \quad S_{ij} = \varepsilon \; \forall i \neq j \\ &S^{-} = \text{diag}_{\oplus}(-s_1, ..., -s_n) \quad \text{s.t.} \quad S \otimes S^{-} = S^{-} \otimes S = I \end{aligned} \tag{3-1}$$

We can then use these arbitrary S when we expand (2-61):

$$\begin{aligned} \overline{\Gamma(N)} &= \max_{i,j} \left[A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)} \right] \\ &= \max_{i,j} \left[S \otimes S^{-} \otimes A^{(\ell_1)} \otimes S \otimes S^{-} \otimes A^{(\ell_2)} \otimes ... \otimes S \otimes S^{-} \otimes A^{(\ell_N)} S \otimes S^{-} \right] \end{aligned} \tag{3-2}$$

But in this form the order of modes still matters, we can however simplify the objective function to make it order independent. We can do this by substituting the matrices with scalars. To get to this scalar form we can substitute:

$$\begin{aligned} \alpha(\ell_m, S) &= \max_{i,j} \left[S^{-} \otimes A^{(\ell)} \otimes S \right] \\ &= \max_{i,j} \left[-s_i + \left[A^{(m)} \right]_{ij} + s_j \right], \quad \forall 1 \leq i, j \leq n \\ \overline{SES^{-}} &= \max_{i,j} \left[s_i - s_j \right], \quad \forall 1 \leq i, j \leq n \end{aligned} \tag{3-3}$$

$$\begin{aligned} \overline{\Gamma(N)} &\leq \max_{i,j} \left[S \otimes \overline{S^{-} A^{(\ell_1)} S} \otimes E \otimes \overline{S^{-} A^{(\ell_2)} S} \otimes E ... E \otimes \overline{S^{-} A^{(\ell_N)} S} \otimes E \otimes S^{-} \right] \\ &= \max_{i,j} \left[S \otimes \alpha(\ell_1, S) \otimes E \otimes \alpha(\ell_2, S) \otimes E \otimes ... \otimes E \otimes \alpha(\ell_N, S) \otimes E \otimes S^{-} \right] \\ &\text{Since } \alpha(\ell, S) \text{is just a one value scalar we can move it out of the max statement.} \\ &= \max_{i,j} \left[S \otimes E \otimes ... \otimes E \otimes S^{-} \right] + \alpha(\ell_1, S) + \alpha(\ell_2, S) + ... + \alpha(\ell_N, S) \\ &= \overline{SES^{-}} + \sum_{k=1}^{N} \alpha(\ell_k, S) \end{aligned} \tag{3-4}$$

Example 3-0.1. Information loss

$$
\begin{aligned}
f(A,S) &= \max\left[S^- \otimes A^{(\ell_1)} \otimes S \otimes S^- \otimes A^{(\ell_2)} \otimes S\right] \\
&= \max\left[\begin{bmatrix} s_1 & \varepsilon \\ \varepsilon & s_2 \end{bmatrix} \otimes \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \otimes \begin{bmatrix} -s_1 & \varepsilon \\ \varepsilon & -s_2 \end{bmatrix} \otimes \begin{bmatrix} s_1 & \varepsilon \\ \varepsilon & s_2 \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \otimes \begin{bmatrix} -s_1 & \varepsilon \\ \varepsilon & -s_2 \end{bmatrix}\right] \\
&= \max\left[\begin{bmatrix} a_{11} & a_{12}+s_1-s_2 \\ a_{21}-s_1+s_2 & a_{22} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12}+s_1-s_2 \\ b_{21}-s_1+s_2 & b_{22} \end{bmatrix}\right] \\
&= \max\begin{bmatrix} a_{11}+b_{11} \oplus a_{12}+b_{21} & a_{11}+b_{12}+s_1-s_2 \oplus a_{12}+b_{22}+s_1-s_2 \\ a_{21}+b_{11}-s_1+s_2 \oplus a_{22}+b_{21}-s_1+s_2 & a_{21}+b_{12} \oplus a_{22}+b_{22} \end{bmatrix} \\
&= \max[a_{11}+b_{11}, a_{12}+b_{21}, a_{21}+b_{12}, a_{22}+b_{22}, a_{11}+b_{12}+s_1-s_2 \\
&\qquad , a_{12}+b_{22}+s_1-s_2, a_{21}+b_{11}-s_1+s_2, a_{22}+b_{21}-s_1+s_2]
\end{aligned}
$$

We can then make this order independent by splitting up the max statement through substituting: $\max_{i,j}\left[S^- \otimes A^{(\ell)} \otimes S\right] = \alpha(\ell, S)$

$$
\begin{aligned}
f(A,S) &= \max\left[S \otimes A^{(\ell_1)} \otimes S^- \otimes S \otimes A^{(\ell_2)} \otimes S^-\right] \\
&\leq \max\left[S \otimes A^{(\ell_1)} \otimes S^-\right] + \max\left[S \otimes A^{(\ell_2)} \otimes S^-\right] \\
&= \max[a_{11}, a_{22}, a_{12}+s_1-s_2, a_{21}-s_1+s_2] + \max[b_{11}, b_{22}, b_{12}+s_1-s_2, b_{21}-s_1+s_2] \\
&= \max[a_{11}+b_{11}, a_{11}+b_{22}, a_{12}+b_{21}, a_{21}+b_{12}, a_{22}+b_{11}, a_{22}+b_{22}, a_{11}+b_{12}+s_1-s_2 \\
&\qquad a_{11}+b_{21}-s_1+s_2, a_{12}+b_{11}+s_1-s_2, a_{12}+b_{12}+2s_1-2s_2 \\
&\qquad a_{12}+b_{22}+s_1-s_2, a_{21}+b_{11}-s_1+s_2, a_{21}+b_{21}-2s_1+2s_2 \\
&\qquad a_{21}+b_{22}-s_1+s_2, a_{22}+b_{12}+s_1-s_2, a_{22}+b_{21}-s_1+s_2]
\end{aligned}
$$

Information is lost in this simplification step as the second max statement has more than double the entries.

In the case of an Switching Max-Plus-Linear (SMPL) system with random switching we can write an expectation of $\alpha(\ell, S)$ if we know the odds of randomly switching into any particular mode $m \in \mathcal{L}$. The odds are denoted $\mathbb{P}[\ell(k) = m] = p_m$ such that $\sum_{m=1}^{L} p_m = 1$.

$$
\mathbb{E}\left[\alpha(\ell, S)\right] = \sum_{m=1}^{L} \alpha(m, S) \cdot p_m \tag{3-5}
$$

We can use the expectation of $\alpha(\ell, S)$ to write an expectation for $\overline{\Gamma(N)}$

$$
\begin{aligned}
\mathbb{E}\left[\overline{\Gamma(N)}\right] &\leq \mathbb{E}\left[\overline{SES^-} + \alpha(\ell_1, S) + \alpha(\ell_2, S) + ... + \alpha(\ell_N, S)\right] \\
&= \overline{SES^-} + \mathbb{E}\left[\alpha(\ell_1, S)\right] + \mathbb{E}\left[\alpha(\ell_2, S)\right] + ... + \mathbb{E}\left[\alpha(\ell_N, S)\right] \\
&= \overline{SES^-} + N \cdot \sum_{m=1}^{L} \alpha(m, S) \cdot p_m
\end{aligned} \tag{3-6}
$$

Alternatively we could write the same thing using conventional algebra in example $\overline{SES^-} = \max_{i,j}[s_i - s_j]$. However for every $\max_{i,j}\left[S^- \otimes A^{(\ell)} \otimes S\right] = \alpha(\ell, S)$ we need to cycle through i and j independently. To denote this independent cycling through all possible permutations of these max statements trough i and j we use two arbitrary sets $\mathcal{I} = \{i_0, i_1, ..., i_L\}$ and $\mathcal{J} = \{j_0, j_1, ..., j_L\}$.

$$\begin{aligned}\mathbb{E}\left[\overline{\Gamma(N)}\right] &\leq \max_{i_0,j_0}[s_{i_0} - s_{j_0}] + N \cdot \sum_{m=1}^{L} p_m \cdot \max_{i_m,j_m}\left[-s_{i_m} + \left[A^{(m)}\right]_{i_m j_m} + s_{j_m}\right] \\ &= \max_{\mathcal{I},\mathcal{J}}\left[s_{i_0} - s_{j_0} + N \cdot \sum_{m=1}^{L} p_m \cdot \left(-s_{i_m} + \left[A^{(m)}\right]_{i_m j_m} + s_{j_m}\right)\right]\end{aligned} \tag{3-7}$$

We have already proven that we can pick any values for the diagonal of S and the result will still be an upperbound on $\mathbb{E}\left[\overline{\Gamma(N)}\right]$. However if we were to just pick some completely arbitrary S we don't know if the resulting upperbound is much higher or quite close to the actual $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ and therefore won't help us much. But since we know that the last line of (3-6) and (3-7) are always larger or equal to $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ we can approach $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ by minimizing it over the arbitrary matrix S or it's diagonal elements $\{s_1, ..., s_n\}$. Which is a minimization problem that can be defined:

$$\min_{S}\left[\overline{SES^-} + N \cdot \sum_{m=1}^{L} \alpha(m, S) \cdot p_m\right]$$

Or using conventional notation: (3-8)

$$\min_{s_1,...,s_n}\left[\max_{\mathcal{I},\mathcal{J}}\left[s_{i_0} - s_{j_0} + N \cdot \sum_{m=1}^{L} p_m \cdot \left(-s_{i_m} + \left[A^{(m)}\right]_{i_m j_m} + s_{j_m}\right)\right]\right]$$

3-1 Approaching $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ with Sequential Quadratic Programming

Another method for attempting to find some S that minimizes (3-8) would be to use type of nonlinear programming. A crude and slow attempt can be made using a genetic algorithm or any other method that only requires no further information about the objective function. Alternatively one can apply sequential quadratic programming, but this requires both the gradient function and hessian to be known. A gradient function is proposed in: (**??**) but for numerical purposes this definition can be improved upon. Problems start when we try to convert the maximum statement in to a differentiable form:

$$\max(a, b) = \lim_{p\to\infty} \sqrt[p]{a^p + b^p}$$

When using a numerical approximation the best we can do is just pick a large p. However if p is uneven and b is a negative value such that $a < |b|$ then $\sqrt[p]{a^p + b^p} < 0 \ll a$ which is far from correct as in that situation $\max(a, b) = a$. Additionally for the same a and b but with the added condition $p \geq 3$ we find that $\mathfrak{Im}\left[\sqrt[p]{a^p + b^p}\right] > 0$ which is definitely unwanted.

These problems can seemingly be resolved by just always picking an even p. However if we pick an even p we run into a new problem where if b is a negative value such that $a < |b|$ then $|b| < \sqrt[p]{a^p + b^p} \gg a$ which is also incorrect.

Luckily we know that in our case the objective function (3-8) should never take on a negative value. As a negative total sorting time implies that our hypothetical package-sorter finishes sorting packages before it starts. Using this property we can rewrite the maximum statement:

$$\begin{aligned} \text{If: } & f(s) = \max[m_q(s), ..., m_q(s)] \geq 0, \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \\ \text{Then: } & f(s) = \max\left[\max[m_q(s), 0], ..., \max[m_q(s), 0]\right] \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \\ & = \lim_{p \to \infty} \sqrt[p]{\sum_{\mathcal{I},\mathcal{J}} \left(\frac{|m_q(s)| + m_q(s)}{2} \right)^p} \end{aligned}$$

For notation purposes we will use: $n(s, i, j) = \frac{1}{2}\left(|m_q(s)| + m_q(s)\right)$ going forward. If you know that the optimal solution to the objective function always has to be equal or larger than some constant C you can improve the precision of your approximation by subtracting it from your objective function:

$$\begin{aligned} \text{If: } & f(a, b) = \max[a, b] \geq C \geq 0, \quad \forall a, b \\ \text{Then: } & f(a, b) - C = \lim_{p \to \infty} \sqrt[p]{\left(\frac{|a - C| + a - C}{2} \right)^p + \left(\frac{|b - C| + b - C}{2} \right)^p} \forall a, b \end{aligned}$$

This is advantageous as it improves the relative difference between entries:

$$\frac{\max[a - C, 0]}{\max[b - C, 0]} \geq \frac{\max[a, 0]}{\max[b, 0]}, \quad \forall a \geq b \geq C > 0$$

And allows us to choose a greater value for p without running into a floating point overflow error. Both improve the precision with which the numerical approximation of the objective function tracks the shape of the exact objective function. For readability purposes we will assume $C = 0$ going forward. We can calculate the gradient of this new objective function using the relationship:

$$\frac{\partial |f(x)|}{\partial x} = \frac{f(x)}{|f(x)|} \cdot \frac{\partial f(x)}{\partial x} \approx \text{sign}[f(x)] \cdot \frac{\partial f(x)}{\partial x}$$

We use the sign function instead of $\frac{f(x)}{|f(x)|}$ as it is not undefined for $f(x) = 0$ which is mathematically incorrect but more useful for a numerical approximation. This means the numerical approximation of the gradient function described in (**??**) comes out as:

$$
\begin{aligned}
[g(s)]_k &= \frac{\partial f(s)}{\partial s_k} \\
&= \left(\sum_{q\in\mathcal{Q}}\left(\frac{m_q(s)+|m_q(s)|}{2}\right)^p\right)^{\frac{1-p}{p}} \cdot \sum_{q\in\mathcal{Q}}\left(\frac{m_q(s)+|m_q(s)|}{2}\right)^{p-1} \cdot \frac{1}{2} \cdot \frac{\partial\left(m_q(s)+|m_q(s)|\right)}{\partial s_k} \\
&\approx \left(\sum_{q\in\mathcal{Q}}\left(\frac{m_q(s)+|m_q(s)|}{2}\right)^p\right)^{\frac{1-p}{p}} \cdot \sum_{q\in\mathcal{Q}}\left(\frac{m_q(s)+|m_q(s)|}{2}\right)^{p-1} \cdot \frac{1}{2} \cdot \frac{\partial m_q(s)}{\partial s_k} \cdot (1+\text{sign}[m_q(s)])
\end{aligned}
\tag{3-9}
$$

The sign function differentiates as the dirac delta function: $\frac{\partial \text{sign}[x]}{\partial x} = \delta(x)$. However for calculating a numerically useful hessian it is more useful to define it as: $\frac{\partial \text{sign}[x]}{\partial x} \approx 0$. With that simplification it is possible to calculate a numerically usable approximation to the Hessian that is never undefined.

However we know that for all $f(s)$ considered in this thesis $f(s)$ is the maximum over a set of linear sub-functions. This means that the actual gradient of $f(s)$ is defined by a contiguous step-functions with its steps at those places where two or more linear sub-functions define the maximum of $f(s)$. This means that the real hessian of $f(s)$ is defined only by dirac delta functions for every step in the gradient function. For the purposes of non-linear programming these dirac delta spikes in the Hessian are not relevant. As such we can more simply define the hessian as a $\mathbb{R}^{n\times n}$ matrix containing zero for every element.

Example 3-1.1. Finding a numerically useful gradient
Say we start with a function:

$$f(x) = \max\left[35-(x-1)^4, -10x-10, \left(\frac{x}{2}-\frac{1}{2}\right)^3+30\right]$$

Then we can calculate the approximation functions:

$$
\begin{aligned}
f_{simple}(x) &\approx \sqrt[p]{(35-(x-1)^4)^p + (-10x-10)^p + \left(\left(\frac{x}{2}-\frac{1}{2}\right)^3+30\right)^p} \\
f_{improved}(x) &\approx \left(\left(-5x+\frac{|10x+10|}{2}-5\right)^p + \left(\frac{\left(\frac{x}{2}-\frac{1}{2}\right)^3}{2} + \frac{\left|\left(\frac{x}{2}-\frac{1}{2}\right)^3+30\right|}{2}+15\right)^p \right. \\
&\quad \left. + \left(-\frac{(x-1)^4}{2} + \frac{\left|(x-1)^4-35\right|}{2} + \frac{35}{2}\right)^p\right)^{\frac{1}{p}}
\end{aligned}
$$

Which when plotted allows us to see the problems with the simple approximation method. In Figure 3-1 you can see how the simplified method fails. In the case of this example this is mostly due to $35-(x-1)^4$ becoming deeply negative.

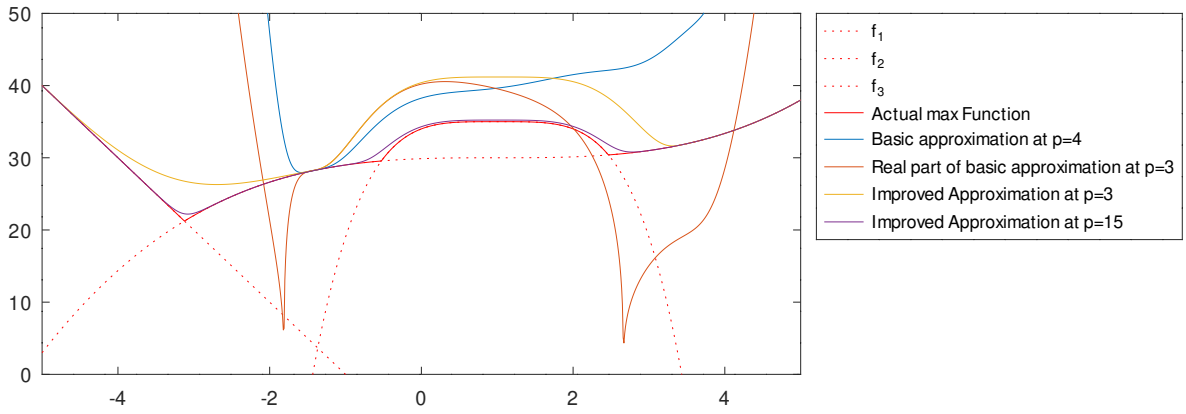

Figure 3-1

Using a subgradient

Due to the nature of the maximum function it is however not necessary to calculate one large differentiable approximation function and derive its gradient. If we expand (3-8) to a single maximum function with $r = (n^2)^{L+1} = n^{2L+2}$ elements we get:

$$f(s) = \max\left[f_1(s), f_2(s), ..., f_r(s)\right] \tag{3-10}$$

For most S the entire maximum function is defined by just one of its elements f_m. This means that in order to find the gradient at these points we need only consider the gradient of f_m:

$$g(s) = \frac{\partial f_m(s)}{\partial s}, \quad \forall f(s) = f_m(s) \neq \{f_1(s), ..., f_{m-1}(s), f_{m+1}(s), ..., f_n(s)\} \tag{3-11}$$

However there are also places where two or more elements of the maximum function have equal values that are the highest out of all elements. If these $f_{m_1}(s)$ and $f_{m_2}(s)$ do not have equal gradients where $f_{m_1}(s) = f_{m_2}(s) = f(s)$ then that leaves the gradient of $f(s)$ undefined. But we can define boundaries on the gradient of $f(s)$ as it clearly should never be steeper than the steepest positive and steepest negative slope defined by all elements that define the maximum function in that point.

$$\frac{\partial f_{m_1}(s)}{\partial s} \leq g(s) \leq \frac{\partial f_{m_2}(s)}{\partial s}, \quad \forall f_{m_1}(s) = f_{m_2}(s) = f(s) \tag{3-12}$$

Example 3-1.2. Subgradient for simple system
Let us define a system: $f(x) = \max\left[-8x - 20, -x, 0.5x, x, 2x - 2\right]$. Then for say $x = 1.5$ we have no issues as $f(1.5) = \max[-32, -1.5, 0.75, 1.5, 1] = 1.5$ has only one element defining $f(x)$ in this case $f_m(x) = x$ so the gradient of $f(x)$ is 1 for $x = 1.5$. However at $x = 0$ we find $f(0) = \max[-20, 0, 0, 0, -2]$ as three elements define $f(x)$ we have an undefined gradient. The three gradients that will span the boundary are $-x \to -1$, $0.5x \to 0.5$, $x \to 1$ which means we are free to pick any $g(0)$ so long as: $-1 \leq g(0) \leq 1$ as can be seen in Figure 3-2.

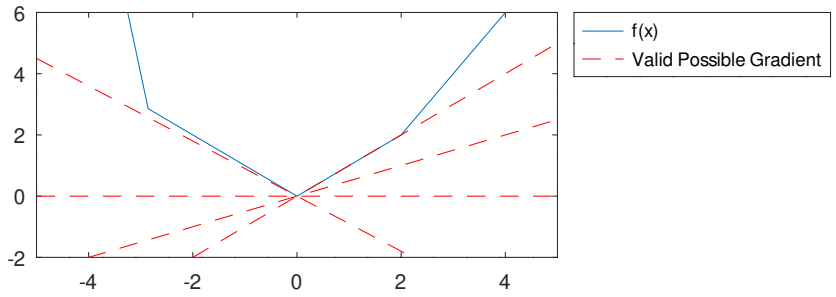

Figure 3-2: Some possible gradients for $f(x)$ in $x = 0$

Since we have freedom to choose the gradient for every intersection of $f(x)$ so long as it stays between it's predecessor and successor we can simply always pick the upper boundary of that range. This would result in the gradient working out as:

$$g(x) = -8 + 7u(x + \frac{20}{7}) + 2u(x) + u(x-2) = \begin{cases} -8, & x < -\frac{20}{7} \\ -1, & -\frac{20}{7} \leq x < 0 \\ 1, & 0 \leq x < 2 \\ 2, & 2 < x \end{cases}$$

This means the Hessian would work out as $H(x) = 7\delta(x + \frac{20}{7}) + 2\delta(x) + \delta(x-2)$. However since we use a numerical approach no algorithm will ever hit those dirac delta function spikes exactly, neither do they provide any useful extra information to any nonlinear programming algorithm. Therefore only in the context of this particular optimization problem we can define $H(x) \approx 0$.

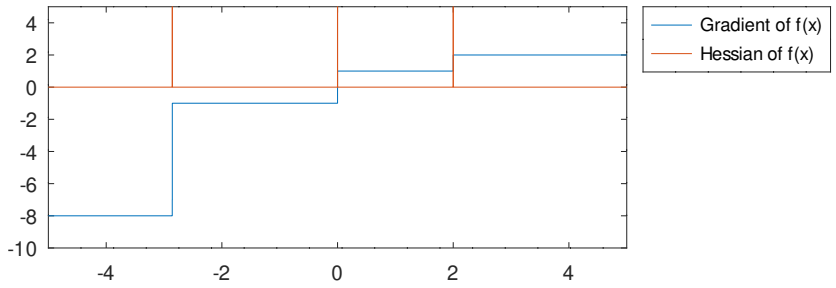

Figure 3-3: Gradient function $g(x)$ of $f(x)$

Since $f_1(s)$ through $f_r(s)$ are all linear we can rewrite them as:

$$f_h(s) = c_{h,0} + c_{h,1} \cdot s_1 + ... + c_{h,n} \cdot s_n$$

Finding the gradient of any $f_h(s)$ does not require any calculation as it is just:

$$g_h(s) = \begin{bmatrix} c_{h,1} & c_{h,2} & ... & c_{h,n} \end{bmatrix}^T$$

The goal then is to find a way to first find which $f_h(s)$ defines $f(s)$ for that particular set of s and then return the corresponding gradient $g_h(s)$. Since this function will be called many

times during sqp optimization it needs to be efficient. Since we use the Octave mathematics environment this is best achieved by vectorizing the problem so that BLAS can be used for calculations. This means finding a way to rewriting (3-10) to the maximum of a single matrix vector product. Then the index for which the vector matrix product has its maximum value can be used to find the gradient.

$$g(s) = \begin{bmatrix} c_{i,1} \\ c_{i,2} \\ \vdots \\ c_{i,n} \end{bmatrix} \quad \text{Such that:} \quad \max_i \left[C \cdot v\right]_i \tag{3-13}$$

It is simplest to define v as a collumn vector starting with 1 followed by all elements on the diagonal of S: $v = \begin{bmatrix} 1 & s_1 & s_2 & s_3 & ... & s_n \end{bmatrix}^T$. Then we define C such that each row multiplied by v results in a unique $f_1(s)$ through $f_r(s)$. We observe that every element of $S^- \otimes A \otimes S$ can be defined as: $A_{ij}^{(m)} - s_i + s_j$. for $S \otimes E \otimes S^-$ the pattern it the same but negative: $0 + s_i - s_j$. This pattern allows us to construct a matrix $D \in \mathbb{R}^{n \times n^2}$

$$D = \begin{bmatrix} 1_1 & 0 & ... & \\ 1_2 & 0 & ... & \\ \vdots & \vdots & \ddots & \\ 1_n & 0 & ... & \\ 0 & 1_1 & ... & \\ \vdots & \vdots & \ddots & \\ 0 & 1_n & ... & \\ \vdots & \vdots & \ddots & \\ 0 & 0 & ... & 1_1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & 1_n \end{bmatrix} - \begin{bmatrix} 1_1 & 0 & 0 & ... & 0 \\ 0 & 1_1 & 0 & ... & 0 \\ 0 & 0 & 1_1 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & ... & 1_1 \\ 1_2 & 0 & 0 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1_n & 0 & 0 & ... & 0 \\ 0 & 1_n & 0 & ... & 0 \\ 0 & 0 & 1_n & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & ... & 1_n \end{bmatrix}$$

Then using this matrix D we can construct D_{SES} such that: $\overline{S \otimes E \otimes S^-} = \max\left[D_{SES} \cdot v\right]$. Then reshaping $A^{(m)}$ to a column vector with the rows of $-D$ matching the elements of A such that $A_{ij}^{(m)} - s_i + s_j$ is still observed. We can create L variants of $D_{SAS}^{(m)}$ such that: $\overline{S^- \otimes A^{(m)} \otimes S} = \max\left[D_{SAS}^{(m)} \cdot v\right]$.

$$D_{SES} = \begin{bmatrix} 0 & \\ \vdots & D \\ 0 & \end{bmatrix}, \quad D_{SAS}^{(m)} = N \cdot p_m \cdot \begin{bmatrix} A_{1,1}^{(m)} & \\ A_{1,2}^{(m)} & \\ \vdots & \\ A_{1,n}^{(m)} & -D \\ A_{2,1}^{(m)} & \\ A_{2,2}^{(m)} & \\ \vdots & \\ A_{n,n}^{(m)} & \end{bmatrix}$$

Using D_{SES} and every mode of D_{SAS} we can now calculate C by summing every possible permutation of the rows of D_{SES} through $D_{SAS}^{(L)}$ which gets us a row for each possible $f_1(s)$ through $f_r(s)$. Therefore if all elements of $A > \varepsilon$ for all modes C has $r = n^{2L+2}$ rows. However if A is sparsely populated for one or more modes the size of C can be decreased. This can be done by removing all rows from every $D_{SAS}^{(m)}$ for which the first column is equal to ε. This can be done since summing any real value with ε returns ε and any row in C containing ε in the first column returns ε when multiplied by v. With less rows in $D_{SAS}^{(m)}$ the number of possible permutations decreases factorially.

$$C = \begin{bmatrix} c_{1,0} & c_{1,1} & \dots & c_{1,n} \\ c_{2,0} & c_{2,1} & \dots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r,0} & c_{r,1} & \dots & c_{r,n} \end{bmatrix} = \begin{bmatrix} [D_{SES}]_{1,*} + \left[D_{SAS}^{(1)}\right]_{1,*} + \dots + \left[D_{SAS}^{(L)}\right]_{1,*} \\ [D_{SES}]_{2,*} + \left[D_{SAS}^{(1)}\right]_{1,*} + \dots + \left[D_{SAS}^{(L)}\right]_{1,*} \\ [D_{SES}]_{1,*} + \left[D_{SAS}^{(1)}\right]_{2,*} + \dots + \left[D_{SAS}^{(L)}\right]_{1,*} \\ \vdots \\ [D_{SES}]_{1,*} + \left[D_{SAS}^{(1)}\right]_{n^2,*} + \dots + \left[D_{SAS}^{(L)}\right]_{n^2,*} \end{bmatrix} \tag{3-14}$$

It is worth noting that $\max[C \cdot v]$ is equivalent to the objective function (3-10). This way (3-13) can be used both to replace the symbolic objective function as well as the gradient function. However there is also a way to calculate the sub-gradient without expanding the summed max functions like we do in (3-10). We leave the separate max functions described in (3-8) as they are, and calculate the subgradient for each max statement separately. Since the objective function just consists of maximums of linear functions we can then just sum the sub-gradients of every separate max function. That way all we need is $L + 1$ matrices: $\left\{D_{SES}, D_{SAS}^{(1)}, ..., D_{SAS}^{(L)}\right\} \in \mathbb{R}^{(n+1)\times n}$ to describe the same thing as defined in (3-13).

$$g(s) = [D_{SES}]_{i_0,*} + \sum_{m=0}^{L} \left[D_{SAS}^{(m)}\right]_{i_m,*} = \begin{bmatrix} [D_{SES}]_{i_0,1} + \left[D_{SAS}^{(1)}\right]_{i_1,1} + ... + \left[D_{SAS}^{(L)}\right]_{i_m,1} \\ [D_{SES}]_{i_0,2} + \left[D_{SAS}^{(1)}\right]_{i_1,2} + ... + \left[D_{SAS}^{(L)}\right]_{i_m,2} \\ \vdots \\ [D_{SES}]_{i_0,n} + \left[D_{SAS}^{(1)}\right]_{i_1,n} + ... + \left[D_{SAS}^{(L)}\right]_{i_m,n} \end{bmatrix} \tag{3-15}$$

Such that:

$$\max_{i_0} [D_{SES} \cdot v]_{i_0} + \sum_{m=1}^{L} \max_{i_m} \left[D_{SAS}^{(m)} \cdot v\right]_{i_m}$$

This method for calculating $g(s)$ saves on both memory usage and calculation steps for larger systems but cannot be solved as a simple matrix vector product. It must be noted that this method of calculating s_1 through s_n does not resolve the fact that the inequality is under defined. So the resulting optimal solution can be any possible s_1 so long as it has the right linear ratio to s_2 through s_n.

3-1-1 Calculation Time and Memory Concerns

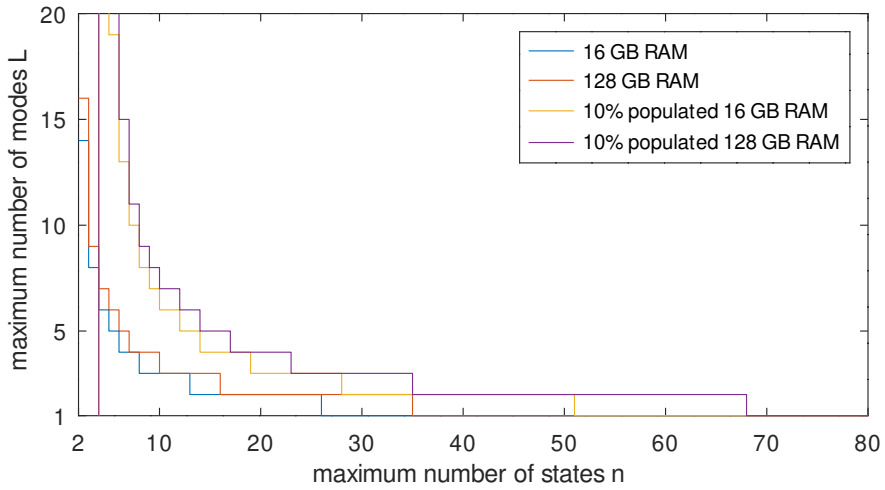

Figure 3-4: Free available RAM required to store C

Since the SQP algorithm itself is not memory intensive it is the objective function and the (sub)gradient function that are the limiting factors. The expanded version of both are most efficiently described by (3-13) which means the size of C is the limiting factor for memory. Defining t_m as the fraction of elements in $A^{(m)} > \varepsilon$ then if every element of C is defined by a double the size of C in bytes can be calculated:

$$(1+n) \cdot n^2 \cdot \prod_{m=1}^{L} t_m \cdot n^2 \leq (1+n) \cdot n^{2+2L} \cdot t_{average}^{L}$$

However if we do not construct the expanded matrix C we can store the gradient- and objective function more compactly as is described in (3-15). Storing D_{SES} and $D_{SAS}^{(1)}$ through $D_{SAS}^{(L)}$

as arrays of doubles would take: $2 \cdot n^2 \cdot (n+1) \cdot (L+1)$ bytes. This could be less if some rows of $D_{SAS}^{(1)}$ through $D_{SAS}^{(L)}$ can be removed because some elements of $A^{(m)} = \varepsilon$. The downside is that (3-15) can not be reduced to the maximum of one BLAS-2 operation like is possible with (3-13). This means that subsequent calculations of the objective- and gradient function can not be as easily or efficiently pipelined. Nevertheless the decrease in calculation steps and memory requirements should be worth it for all but the smallest systems.

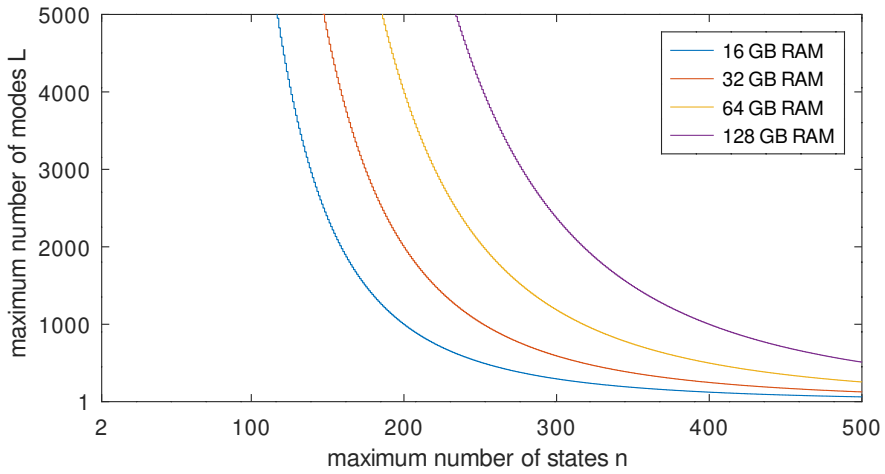

Figure 3-5: Free available RAM required to store D_{SES} and $D_{SAS}^{(1)}$ through $D_{SAS}^{(L)}$

3-2 Approaching $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ with Linear Programming

In order to minimize optimization function (3-8) using linear programming we again define a column vector $s \in \mathbb{R}^n$ with:

$$s_m = [S]_{mm}$$

Which allows us to rewrite equation (3-8) as a linear programming problem.

$$\begin{aligned} &\min_s \sigma_0 + N \cdot \sum_{m=1}^{L} \sigma_m \cdot p_m \\ &\text{s.t.} \\ &\sigma_0 \geq \overline{SES^-} = s_i - s_j, \ \forall i,j \\ &\sigma_m \geq \alpha(m,S), \ \forall m = -s_i + \left[A^{(m)}\right]_{ij} + s_j, \ \forall i,j,m \end{aligned} \tag{3-16}$$

In order to apply the simplex algorithm we need to restate the problem in the form:

$$\begin{aligned} &\min_z C \cdot z \\ &\text{such that:} \\ &E \cdot z \geq b \end{aligned}$$

And even though the optimization variable s only appears in the boundary equation and not the objective function, there is a way to rewrite (3-16). This done by just having both σ and

s in the optimization variable z. This is possible since you can just disregard s by setting $\{c_{L+2}, ..., c_{L+1+n}\} = 0$.

$$C = \begin{bmatrix} 1 & Np_1 & ... & Np_L & 0_1 & 0_2 & ... & 0_n \end{bmatrix}$$
$$z = \begin{bmatrix} \sigma_0 & \sigma_1 & ... & \sigma_L & s_1 & s_2 & ... & s_n \end{bmatrix}^T$$

With x in this form we can also easily rewrite the boundary conditions by moving both σ and s to the same side of the inequality. Starting with the first boundary condition concerning σ_0:

$$\sigma_0 \geq s_i - s_j, \quad \forall i, j$$
$$\sigma_0 - s_i + s_j \geq 0, \quad \forall i, j$$

Since the boundary condition is true for all i and j one part of the final A matrix needs to contain a row for every possible permutation of that statement. Since i and j cycle independently that is $n \cdot n$ permutations. However a number of these permutations are superfluous. There are n cases of $i = j$ which brings the number of relevant permutations to $n^2 - n + 1$. For the sake of clarity these $n - 1$ superfluous rows of E_1 are not left out below:

$$E_1 = \begin{bmatrix} 1 & 0_1 & ... & 0_L & -1+1 & 0_2 & ... & 0_n \\ 1 & 0_1 & ... & 0_L & -1 & 1 & ... & 0_n \\ 1 & 0_1 & ... & 0_L & 1 & -1 & ... & 0_n \\ 1 & 0_1 & ... & 0_L & 0_1 & -1+1 & ... & 0_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0_1 & ... & 0_L & 0_1 & 0_2 & ... & -1+1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Then moving on to the second set of boundary conditions concerning $\{\sigma_1, ..., \sigma_L\}$. Again we can move all σ and s to one side of the inequality leaving the known elements of the SMPL system's A on the other:

$$\sigma_m \geq -s_i + \left[A^{(m)}\right]_{ij} + s_j, \quad \forall i, j, m$$
$$\sigma_m + s_i - s_j \geq \left[A^{(m)}\right]_{ij}, \quad \forall i, j, m$$

Again we have to cycle through all permutations of i and j, but now all modes m as well which means $n^2 \cdot L$ permutations. This time all $i = j$ after the first are not necessarily superfluous as $\sigma_m \geq \left[A^{(m)}\right]_{ii}$ still needs to be met. But if one wanted to minimize the number of rows in A_2 at any cost one could search for $\max_i \left[A^{(m)}\right]_{ii}$ for every mode and leave out all other elements on the diagonal of $A^{(m)}$. Doing so would decrease the number of rows in A_2 by $L \cdot (n-1)$. Also since any number in $\mathbb{R}$ is always greater than ε and $\sigma_m + s_i - s_j \in \mathbb{R}$, $\forall i, j, m$ all rows for which $\left[A^{(m)}\right]_{ij} = \varepsilon$ are also superfluous. Accounting for this fact reduces the number of rows in E_2 by the sum total of ε elements in $A^{(m)}$ across all modes of the SMPL system.

$$E_2 = \begin{bmatrix} 0_0 & 1 & 0_2 & ... & 0_L & 1-1 & 0_2 & ... & 0_n \\ 0_0 & 1 & 0_2 & ... & 0_L & 1 & -1 & ... & 0_n \\ 0_0 & 1 & 0_2 & ... & 0_L & -1 & 1 & ... & 0_n \\ 0_0 & 1 & 0_2 & ... & 0_L & 0_1 & 1-1 & ... & 0_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_0 & 1 & 0_2 & ... & 0_L & 0_1 & 0_2 & ... & 1-1 \\ 0_0 & 0_1 & 1 & ... & 0_L & 1-1 & 0_2 & ... & 0_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_0 & 0_1 & 1 & ... & 0_L & 0_1 & 0_2 & ... & 1-1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_0 & 0_1 & 0_2 & ... & 1 & 0_1 & 0_2 & ... & 1-1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} \left[A^{(1)}\right]_{11} \\ \left[A^{(1)}\right]_{12} \\ \left[A^{(1)}\right]_{21} \\ \left[A^{(1)}\right]_{22} \\ \vdots \\ \left[A^{(1)}\right]_{nn} \\ \left[A^{(2)}\right]_{11} \\ \vdots \\ \left[A^{(2)}\right]_{nn} \\ \vdots \\ \left[A^{(L)}\right]_{nn} \end{bmatrix}$$

However this leaves z under defined as E_1 and E_2 only constrain $s_1, ..., s_n$ relative to each other, leaving one degree of freedom. In order to reduce the infinite number of combinations of $s_1, ..., s_n$ that can fulfill this constraint we can add one arbitrary constraint. The simplest would be to just set an arbitrary element of s equal to zero say $s_1 = 0$. But for future purposes it is more useful to set the sum of s equal to zero as this will allow us to more easily see how far the different elements of s deviate from the mean, which would in this case be 0. This means we have to introduce one more row to E:

$$E_{equal} = \begin{bmatrix} 0_0 & 0_1 & ... & 0_L & 1_1 & 1_2 & ... & 1_n \end{bmatrix}, \quad b_{equal} = 0$$

Most linear programming libraries allow for you to define the type of constraint by row of the constraint matrix. For our purposes we can set the first row to force equality and all rows after to greater than or equal.

$$E = \begin{bmatrix} E_{equal} \\ E_1 \\ E_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_{equal} \\ b_1 \\ b_2 \end{bmatrix}$$

3-2-1 Memory Concerns

The memory bottleneck in the case of this optimization problem will be the E matrix in all non-trivial cases. The height of E_1 is $n^2 - n$ as the diagonal of $\overline{SES^-}$ does not contain any useful information. The height of E_2 is $L \cdot n^2$ bringing the height of E to $n^2 \cdot (1 - \frac{1}{n} + L)$. The width of E is equal to the number of items in the x vector: $1 + L + n$. Then assuming the elements of E are stored as floating point doubles the size of the array in bytes would be:

$$2 \cdot n^2 \cdot \left(1 - \frac{1}{n} + L\right) \cdot (1 + n + L) = 2L^2n^2 + 2Ln^3 + 4Ln^2 - 2Ln + 2n^3 - 2n$$

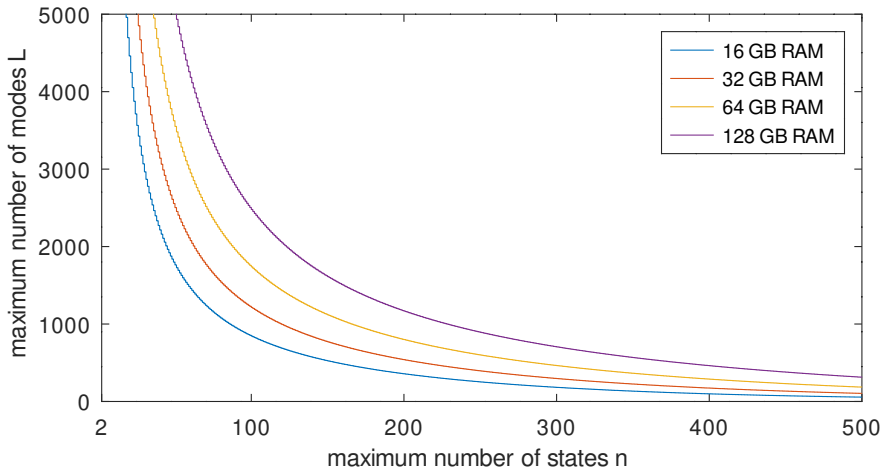

Figure 3-6: Free available RAM required to store E

As can be seen in Figure 3-6 n is the more limiting factor as the size of the E matrix always grows at a slower rate than max $[\mathcal{O}(L^2n^2), \mathcal{O}(Ln^3)]$. For a system 2 modes we can find the optimal solution for at most $n = 2200$ if we have 64 gigabytes of RAM available. Using the same amount of RAM we can apply this method to a system of size $n = 2$ with at most $L = 89440$ modes again showing the number of modes to be less limiting than the size of A.

Chapter 4

Estimating $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ using a Marginal Cost Model

The simplest method for finding an upper bound on $\mathbb{E}[\overline{\Gamma(N)}]$ is to just calculate:

$$\mathbb{E}\left[\overline{\Gamma(N)}\right] \leq N \cdot \sum_{m=1}^{L} p_m \cdot \overline{A^{(m)}}$$

This method can be slightly improved by recognizing that there is a way to get closer to $\mathbb{E}[\overline{\Gamma(N)}]$ since in some Switching Max-Plus-Linear (SMPL) systems a new cycle needn't always wait at the same bottleneck if the mode switches. We could write something similar to this using a mode specific delay d_m. In this simple first order approximation d_m will be somewhere between the actual eigenvalue of the mode or marginal processing time of the mode and the largest element in $A^{(m)}$ or $\lambda_m \leq d_m \leq \overline{A^{(m)}}$ such that we can still write:

$$\mathbb{E}\left[\overline{\Gamma(N)}\right] \leq d_0 + N \cdot \sum_{m=1}^{L} p_m \cdot d_m \tag{4-1}$$

With d_0 the start-up cost if the bottlenecking cycle can contain more than one package. As an example: in the case of a duplex printer multiple pages are in different stages within the bottlenecking print cycle at the same time. Thus when starting up this whole cycle needs to be filled which takes more time than the marginal delay per sheet of paper after this initial start-up cost. This marginal delay is then denoted as $d_m \approx \overline{A^{(m)}}$ it must be noted that this approximate relationship only holds for a first order approximation. To get as close to $\mathbb{E}[\overline{\Gamma(N)}]$ as possible we want to pick the smallest values d for which (4-1) is still valid. This can be written as a linear programming problem where you minimize:

$$\min_{d} d_0 + N \cdot \sum_{m=1}^{L} p_m \cdot d_m$$

With the boundary condition that no N length permutation of $\max\left[A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)}\right]$ should ever be larger than $d_0 + d_{\ell_1} + ... + d_{\ell_N}$. Critically the values of d obtained through this method are only guaranteed to give you an upperbound on $\mathbb{E}[\overline{\Gamma(N)}]$ for exactly the N at which the minimization was done. As for example $\overline{A^{(1)} \otimes A^{(1)} \otimes A^{(1)}} \cdot \frac{2}{3}$ needn't be equal to $\overline{A^{(1)} \otimes A^{(1)}}$. Because if the bottleneck is a circuit containing more than one edge $c_1 \to ... \to c_n \to c_1$ and not all of these edges have equal delay we find that:

$$\overline{\bigotimes^{N} A^{(1)}} - \overline{\bigotimes^{N-1} A^{(1)}} \in \{c_1, ..., c_n\}$$

Which means that d_1 will vary depending on the value of N for which the minimization is done. However as N increases the d_1 found gets closer to the asymptotic d_1 found if the minimization is performed to the limit $N \to \infty$. The same can be said for all other elements of d.

4-1 Higher Order Approximation

This estimate can be improved by taking more information into account. In example if two modes have mostly independent paths through your system it can be much more efficient to switch between these two modes rather than continuously working through one mode and then the other. This kind of savings can't be shown in a first order approximation as it does not consider switching costs. However we can introduce d_{ℓ_{k-1},ℓ_k} denoting the marginal cost or savings for switching from ℓ_{k-1} to ℓ_k. Meaning a marginal cost for $d_{\ell_{k-1},\ell_k} > 0$ and a marginal savings for $d_{\ell_{k-1},\ell_k} < 0$ in a case where $d_{\ell_{k-1},\ell_k} = 0$ both modes presumably share the same bottleneck. This means the boundary condition now writes as: $\max_{ij}\left[A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)}\right] \leq d_0 + d_{\ell_1} + d_{\ell_1,\ell_2} + d_{\ell_2} + ... + d_{\ell_{N-1}} + d_{\ell_{N-1},\ell_N} + d_{\ell_N}$.

Introducing this second order switching cost means the minimization works out to:

$$\min_d d_0 + N \cdot \sum_{m=1}^{L} p_m \cdot d_m + (N-1) \cdot \sum_{m_1=1}^{L} \sum_{m_2=1}^{L} p_{m_1} \cdot p_{m_2} \cdot d_{m_1,m_2}$$

Such that: (4-2)

$$\overline{A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)}} \leq d_0 + \sum_{k=1}^{N} d_{\ell_k} + \sum_{k=2}^{N} d_{\ell_{k-1},\ell_k},\ \forall 1 \leq \ell \leq L$$

In order to formulate this into a linear programming problem that we can solve using the simplex algorithm we need to rewrite it to the form:

$$\min C \cdot z$$
$$\text{such that:}$$
$$E \cdot z \geq b$$

Where we define the column vector z containing all marginal delays, and weighting row vector C such that $C \cdot z$ is equivalent to the minimization function in (4-2). The size of z and C

depends on the order of the approximation which can be calculated: $1 + \sum_{i=1}^{Order} L^i$. We use a second order approximation which means $z \in \mathbb{R}^{1+L+L^2 \times 1}$ and $C \in \mathbb{R}^{1 \times 1+L+L^2}$.

$$z = \begin{bmatrix} d_0 & d_1 & ... & d_L & d_{1,1} & d_{1,2} & ... & d_{2,1} & d_{2,2} & ... & d_{L,L} \end{bmatrix}^T$$
$$C = \begin{bmatrix} 1 & Np_1 & ... & Np_L & (N-1)p_1p_1 & (N-1)p_1p_2 & ... & (N-1)p_2p_1 & (N-1)p_2p_2 & ... & (N-1)p_Lp_L \end{bmatrix}$$

Similarly we have to rewrite the boundary conditions of (4-2) to a matrix vector product $E \cdot z$ that is always greater or equal than the vector b. We can find b by calculating every permutation of $\max_{ij} \left[A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)} \right]$. That is L^N permutations which means $b \in \mathbb{R}^{L^N \times 1}$. Then every element in any row of E denotes the number of marginal delays $\{d_0,\ d_1,\ ...,\ d_L,\ d_{1,1},\ ...,\ d_{L,L}\}$ involved for the corresponding permutation of $\max_{ij} \left[A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)} \right]$. This means that for any row of E the first element should always be 1 since d_0 is always involved. Then element 2 through $1 + L$ should sum N since there are N A^{ℓ_k} matrices for every permutation. And element $2 + L$ through $1 + L + L^2$ should sum $N - 1$ since there $N - 1$ spaces between every $A^{\ell_k} \otimes A^{\ell_{k+1}}$ for every permutation. These attributes can be useful to quickly check if the E matrix was constructed correctly. We can also check this by calculating the sums of the columns of E. Trivially the first column should sum L^N since every element of that column should be 1, then column 2 through $1 + L$ should sum $N \cdot L^{N-1}$ and lastly column $2 + L$ through $1 + L + L^2$ should sum $(N-1) \cdot L^{N-2}$.

$$b = \begin{bmatrix} \overline{A^{(1)} \otimes ... \otimes A^{(1)} \otimes A^{(1)}} \\ \overline{A^{(1)} \otimes ... \otimes A^{(1)} \otimes A^{(2)}} \\ \overline{A^{(1)} \otimes ... \otimes A^{(1)} \otimes A^{(3)}} \\ \vdots \\ \overline{A^{(1)} \otimes ... \otimes A^{(1)} \otimes A^{(L)}} \\ \overline{A^{(1)} \otimes ... \otimes A^{(2)} \otimes A^{(1)}} \\ \vdots \\ \overline{A^{(L)} \otimes ... \otimes A^{(L)} \otimes A^{(L-1)}} \\ \overline{A^{(L)} \otimes ... \otimes A^{(L)} \otimes A^{(L)}} \end{bmatrix}, E = \begin{bmatrix} 1 & N & 0 & 0 & ... & 0 & 0 & N-1 & 0 & 0 & ... & 0 & 0 \\ 1 & N-1 & 1 & 0 & ... & 0 & 0 & N-2 & 1 & 0 & ... & 0 & 0 \\ 1 & N-1 & 0 & 1 & ... & 0 & 0 & N-2 & 0 & 1 & ... & 0 & 0 \\ & & & & & & \vdots & & & & & & \\ 1 & N-1 & 0 & 0 & ... & 0 & 1 & N-2 & 0 & 0 & ... & 0 & L \\ 1 & N-1 & 1 & 0 & ... & 0 & 0 & N-3 & 1 & 0 & ... & 0 & 0 \\ & & & & & & \vdots & & & & & & \\ 1 & 0 & 0 & 0 & ... & 1 & N-1 & 0 & 0 & 0 & ... & 1 & N-2 \\ 1 & 0 & 0 & 0 & ... & 0 & N & 0 & 0 & 0 & ... & 0 & N-1 \end{bmatrix} \tag{4-3}$$

Example 4-1.1. Working out the linear optimization matrices for a simple problem
To clarify we will work through a simple example system. Let's say we want to predict three operations out into the future $N = 3$ and the system only has two modes $L = 2$ with unequal odds at occurring:

$$p_1 = 0.7, \quad p_2 = 0.3, \quad A^{(1)} = \begin{bmatrix} 3 & \varepsilon \\ 7 & 9 \end{bmatrix}, \quad A^{(2)} = \begin{bmatrix} 5 & 3 \\ 8 & \varepsilon \end{bmatrix}$$

Then we define z and can calculate all entries to C:

$$\begin{aligned} z &= \begin{bmatrix} d_0 & d_1 & d_2 & d_{1,1} & d_{1,2} & d_{2,1} & d_{2,2} \end{bmatrix}^T \\ C &= \begin{bmatrix} 1 & 3 \cdot 0.7 & 3 \cdot 0.3 & 2 \cdot 0.7^2 & 2 \cdot 0.7 \cdot 0.3 & 2 \cdot 0.3 \cdot 0.7 & 2 \cdot 0.3^2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2.1 & 0.9 & 0.98 & 0.42 & 0.42 & 0.18 \end{bmatrix} \end{aligned}$$

Then since $N = 3$ and $L = 2$ we have to go through $L^N = 8$ permutations which means $b \in \mathbb{R}^{8\times 1}$ and $E \in \mathbb{R}^{8\times 7}$. Then we populate all entries of E as described in (4-3):

$$b = \begin{bmatrix} \overline{A^{(1)} \otimes A^{(1)} \otimes A^{(1)}} \\ \overline{A^{(1)} \otimes A^{(1)} \otimes A^{(2)}} \\ \overline{A^{(1)} \otimes A^{(2)} \otimes A^{(1)}} \\ \overline{A^{(1)} \otimes A^{(2)} \otimes A^{(2)}} \\ \overline{A^{(2)} \otimes A^{(1)} \otimes A^{(1)}} \\ \overline{A^{(2)} \otimes A^{(1)} \otimes A^{(2)}} \\ \overline{A^{(2)} \otimes A^{(2)} \otimes A^{(1)}} \\ \overline{A^{(2)} \otimes A^{(2)} \otimes A^{(2)}} \end{bmatrix} = \begin{bmatrix} 27 \\ 26 \\ 20 \\ 22 \\ 21 \\ 20 \\ 20 \\ 19 \end{bmatrix}, \quad E = \begin{bmatrix} 1 & 3 & 0 & 2 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 2 & 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 2 & 0 & 1 & 1 & 0 \\ 1 & 1 & 2 & 0 & 0 & 1 & 1 \\ 1 & 0 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

4-1-1 Enabeling extrapolation beyond N_{fit}

We should note that since the number of rows in b scales with L^N calculating further out into the future $N > 10$ quickly becomes totally infeasible. However it is possible to rewrite the optimization problem such that a $z_{N_{fit}}$ once fitted can be used to project out beyond the N_fit for which it was fitted. If we were to take the original optimization problem limit $N \to \infty$:

$$\begin{aligned} &\min_d \lim_{N\to\infty} d_0 + N \cdot \sum_{m=1}^{L} p_m \cdot d_m + (N-1) \cdot \sum_{m_1=1}^{L} \sum_{m_2=1}^{L} p_{m_1} \cdot p_{m_2} \cdot d_{m_1,m_2} \\ \approx &\min_d \lim_{N\to\infty} (N-1) \cdot \sum_{m=1}^{L} p_m \cdot d_m + (N-1) \cdot \sum_{m_1=1}^{L} \sum_{m_2=1}^{L} p_{m_1} \cdot p_{m_2} \cdot d_{m_1,m_2} \end{aligned}$$

Clearly at $\lim_{N\to\infty}$ any startup cost defined in d_0 is negligible and $\lim_{N\to\infty} N - 1 \approx \lim_{N\to\infty} N$. This results in a new linear optimization problem:

$$\begin{aligned} &\min_d N \cdot \sum_{m=1}^{L} p_m \cdot d_m + N \cdot \sum_{m_1=1}^{L} \sum_{m_2=1}^{L} p_{m_1} \cdot p_{m_2} \cdot d_{m_1,m_2} \\ &\text{Such that:} \\ &\overline{A^{(\ell_1)} \otimes A^{(\ell_2)} \otimes ... \otimes A^{(\ell_N)}} \leq + \sum_{k=1}^{N} d_{\ell_k} + \frac{N}{N-1} \cdot \sum_{k=2}^{N} d_{\ell_{k-1},\ell_k}, \ \forall 1 \leq \ell \leq L \end{aligned} \tag{4-4}$$

Again we can redefine this problem in matrix form with:

$$\begin{aligned} z &= \begin{bmatrix} d_1 & ... & d_L & d_{1,1} & d_{1,2} & ... & d_{2,1} & d_{2,2} & ... & d_{L,L} \end{bmatrix}^T \\ C &= \begin{bmatrix} Np_1 & ... & Np_L & Np_1p_1 & Np_1p_2 & ... & Np_2p_1 & Np_2p_2 & ... & Np_Lp_L \end{bmatrix} \\ C_{marginal} &= \begin{bmatrix} p_1 & ... & p_L & p_1p_1 & p_1p_2 & ... & p_2p_1 & p_2p_2 & ... & p_Lp_L \end{bmatrix} \end{aligned}$$

It should be noted that all elements of C can be divided by N to calculate $C_{marginal}$ which can then be used for projecting beyond the N_{fit} for which $z_{N_{fit}}$ was fitted. We can calculate an approximation to the expectation on the total processing time $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ for some N simply:

$$\begin{aligned} &\mathbb{E}\left[\overline{\Gamma(N)}\right] \leq N \cdot C_{marginal} \cdot z_{N_{fit}} \\ &\text{with:} \\ &N \geq N_{fit} \end{aligned} \tag{4-5}$$

Then b is calculated in exactly the same way as described in (4-3) however with the differently sized C and z and differently weighted C E also needs to be adjusted. Firstly the first column of the $E_{original}$ described in (4-3) needs to be removed. Then with this first column removed column 1 through L of E_{new} should still sum N for every row, these are thus exactly the same as for $E_{original}$. Assuming a second order approach that leaves us to fill columns $1+L$ through $L+L^2$ in E_{new} which correspond to columns $2+L$ through $1+L+L^2$ in $E_{original}$. However every row of this section of E_{new} should sum N while it sums $N-1$ in $E_{original}$ this is rectified by simply multiplying every element of this section of $E_{original}$ by $\frac{N}{N-1}$ before copying it over to E_{new}. This process can be repeated for higher order approximations, in a third order approximation we would have to copy column $2+L+L^2$ through $1+L+L^2+L^3$ of $E_{original}$ and scale it by $\frac{N}{N-2}$ and so on for fourth order and beyond.

This new calculation overweights the marginal switching costs relative to the marginal mode cost by a factor $\frac{N-1}{N}$ and the lack of d_0

Example 4-1.2. Rewriting to the limit of N to ∞ style fit
To clarify, we take the exact same system as described in the previous example and now want to write z, C, b and E such that they can be used for minimizing (4-4). There are only slight changes to z and C:

$$\begin{aligned} z &= \begin{bmatrix} d_1 & d_2 & d_{1,1} & d_{1,2} & d_{2,1} & d_{2,2} \end{bmatrix}^T \\ C &= \begin{bmatrix} 3 \cdot 0.7 & 3 \cdot 0.3 & 3 \cdot 0.7^2 & 3 \cdot 0.7 \cdot 0.3 & 3 \cdot 0.3 \cdot 0.7 & 3 \cdot 0.3^2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2.1 & 0.9 & 0.98 & 0.42 & 0.42 & 0.18 \end{bmatrix} \end{aligned}$$

Of course one side of the boundary conditions b need not be changed at all. However E does need to be adjusted. The first column that corresponded to d_0 is to be removed as d_0 is also removed from z. And elements in E need to be scaled such that the summed weight of every section is N for every order. This can be achieved by multiplying each element of these sections with with their respective scalar $\frac{N}{N-Order+1}$. In our case that means multiplying the columns corresponding to d_1 and d_2 with $\frac{3}{3-1+1} = 1$ and the columns corresponding to $d_{1,1}$ through $d_{2,2}$ with $\frac{3}{3-2+1} = 1.5$.

$$E = \begin{bmatrix} 3 & 0 & \frac{3}{2}\cdot 2 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 0 \\ 2 & 1 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 0 \\ 2 & 1 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 0 \\ 1 & 2 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 1 \\ 2 & 1 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 0 \\ 1 & 2 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 0 \\ 1 & 2 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 1 & \frac{3}{2}\cdot 1 \\ 0 & 3 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 0 & \frac{3}{2}\cdot 2 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 3 & 0 & 0 & 0 \\ 2 & 1 & 1.5 & 1.5 & 0 & 0 \\ 2 & 1 & 0 & 1.5 & 1.5 & 0 \\ 1 & 2 & 0 & 1.5 & 0 & 1.5 \\ 2 & 1 & 1.5 & 0 & 1.5 & 0 \\ 1 & 2 & 0 & 1.5 & 1.5 & 0 \\ 1 & 2 & 0 & 0 & 1.5 & 1.5 \\ 0 & 3 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Then the question is: L for what N do you need to fit an approximation such that it has good predictive properties? To answer this question we did Monte Carlo simulations using randomly generated systems. None to all elements of every mode could be set to ϵ the values of all non-ϵ elements of the matrices ranged between 0 and 10. The size of the systems ranged $n = 2$ and $n = 6$, and the number of modes ranged $L = 2$ through $L = 5$. The use of such small systems was necessary to run many different Monte Carlo iterations, already indicating the calculation time problem this method quickly runs into when applied to larger systems. But as will be discussed later the results of these simulations can most likely be extrapolated for larger systems.

For every randomly generated system we calculated a $b_{N_{fit}}$ with N_{fit} ranging from 2 to 7. Then fitted a respective $z_{N_{fit}}$ for every respective $b_{N_{fit}}$. Then we simulated each system for 8 steps effectively calculating a b_8. Similarly we generate an E_8 but as we are not fitting but actually predicting there is no need to overweight switching costs to consider the case $\lim_{N\to\infty}$ which means we construct E_8 using the method discussed in (4-3). Then we either remove the first column of this E_8 or add a zero at the start of every fitted $z_{N_{fit}}$. Then we multiply $E_8 \cdot z_{N_{fit}}$ to calculate an estimate for all permutations. We also calculate a vector P_8 containing the odds for every permutation.

$$P_8 = \begin{bmatrix} p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \\ p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_2 \\ \vdots \\ p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_L \\ p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_1 \cdot p_2 \cdot p_1 \\ \vdots \\ p_L \cdot p_L \cdot p_L \cdot p_L \cdot p_L \cdot p_L \cdot p_L \cdot p_L \end{bmatrix} \tag{4-6}$$

This allows us to calculate a weighted absolute error in the following way:

$$Error_{Nfit} = P_8^T \cdot \left(E_8 \cdot z_{N_{fit}} - b_8\right) \tag{4-7}$$

In Figure 4-1 we plot this absolute error out for all N_fit. This is done for 10 randomly generated systems where the error drops asymptotically.

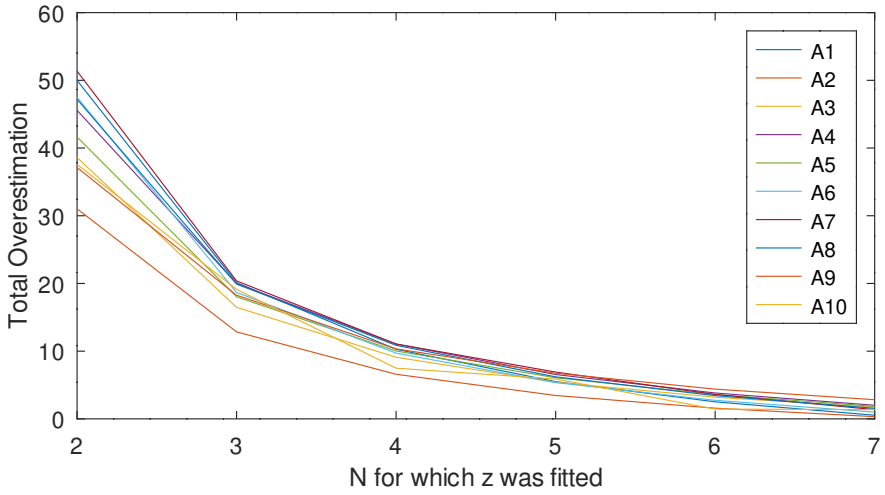

Figure 4-1: Absolute error for every fit on an N=8 simulation

However to generalize these outcomes it is more useful to define a relative error. Ideally one might want to define this error elementwise before weighting. This can be defined using a Hadamard division: $C_{ij} = \frac{A_{ij}}{B_{ij}} \rightarrow C = A \oslash B$

$$RelativeError_{Nfit} = P_8^T \cdot \left(\left(E_8 \cdot z_{N_{fit}} - b_8\right) \oslash b_8\right) \cdot 100 \tag{4-8}$$

This method would probably work for most normal systems, but for these randomly generated systems modes m_{crit} can occur where nearly all elements are set to ε and any elements larger than ε can still be very near to 0. In such cases one element of b_8 for the permutation containing only that one critical mode $[b_8]_{i_{crit}} = \overline{\bigotimes^8 A^{(m_{crit})}}$ will then also be zero or near zero. In such cases the elementwise relative error will explode to near infinity. This does not confer much useful information as the absolute error is usually still small en when we calculate the relative error again removing the critical element $[b_8]_{m_{crit}}$ from b_8 and the corresponding row in E_8 and element in P_8 as well as rescaling this new P_8 such that it still sums 1 by multiplying with the scalar: $\frac{1}{1-p^8_{m_{crit}}}$. Then the result once again clusters with those systems that do not have near zero modes. But in order to still allow for these rare cases in the simulation we calculated relative error after applying weights. Of which the results correlate strongly with the element wise approach for normal systems.

$$RelativeError_{Nfit} = \frac{P_8^T \cdot \left(E_8 \cdot z_{N_{fit}} - b_8\right)}{P_8^T \cdot b_8} \cdot 100 \tag{4-9}$$

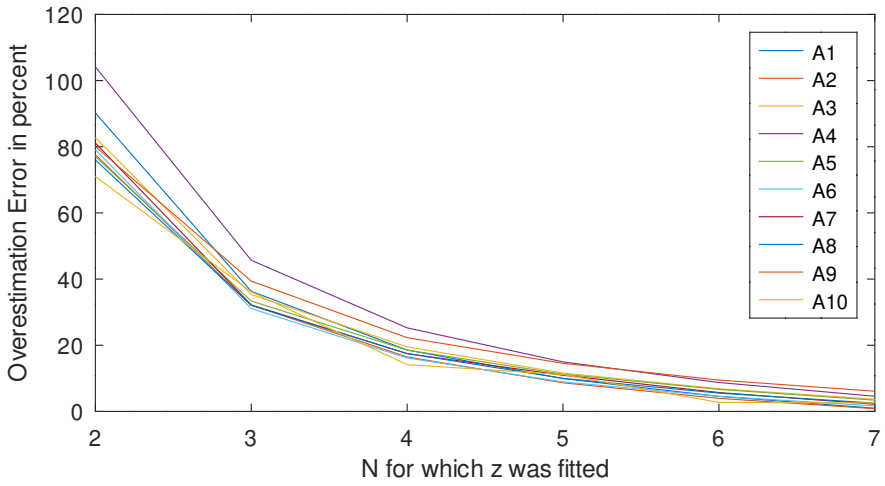

Figure 4-2: Relative Error for every fit on an N=8 simulation

4-2 Problematic systems

It must be noted that in a larger monte carlo simulations we found there were some randomly generated systems for which both the absolute and relative error did not asymptotically decrease to zero. These systems generally had sparse A-matrices with low value entries, but at least two modes for which a mode switch caused a large marginal delay. Which explains why these systems have such high relative error scores to their fairly regular absolute error. In Figure 4-3 and Figure 4-4 we ran a Monte Carlo where we purposefully generated systems with at least 80% of elements set to ε leaving all other variables the same as for Figure 4-1 and Figure 4-2. Clearly this method is unfit for predicting processing time for such systems.

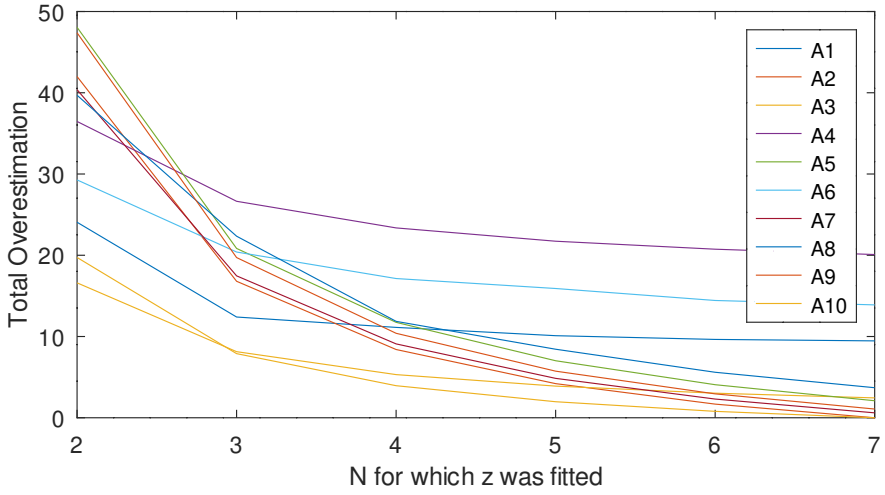

Figure 4-3: Absolute error for every fit on an N=8 simulation

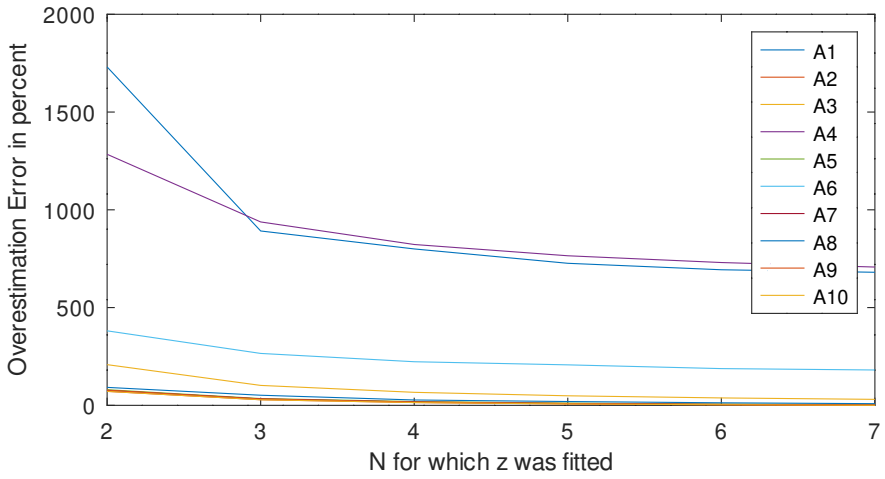

Figure 4-4: Relative Error for every fit on an N=8 simulation

4-3 Dropping the upperbound requirement when estimating $\mathbb{E}\left[\overline{\Gamma(N)}\right]$

Up to this point we have worked under the assumption that our prediction of processing time always needs to be larger or equal to the actual expectation $\mathbb{E}\left[\overline{\Gamma(N)}\right]$. However if this is not a requirement we can improve calculation time and decrease the absolute error on our predictions.

4-3-1 Higher N_{fit} through incomplete b

So far we have always calculated all permutations to find the constraint vector b and constraint matrix E. This means that for systems with a high number of modes L we can only fit z for a relatively small N_{fit} as the number of possible permutations $L^{N_{fit}}$ quickly explodes. But for a reasonable quality fit we needn't have nearly so many constraints as statistically every new permutation contains marginally less new information than the previous as we go through all permutations. This is due to the simple fact that the constraint defines a minimum. If we were to imagine that we were throwing two regular dice and keeping track of the highest number rolled. Say after rolling 20 times the highest total we noted down is 10, for our next roll the odds we find a higher value is just $\frac{1}{12}$ and if we roll an eleven the chance of finding a new even more constraining roll is just $\frac{1}{36}$. If we only needed to know the maximum possible roll at 95% certainty with a 20% error margin we could have stopped rolling at 35 rolls however it would take 107 rolls for a 95% certainty of a 12 showing up. Similarly if we accept some error margin on our estimate of z we needn't have nearly as long a b vector for the same N_{fit}. However knowing ahead of time how many rolls it takes to empirically find the maximum sum of two dice means you already have a working model of these dice. In the case of deciding what is an acceptable length for b we do not have such a model, as this model is what we are trying to find out by solving this minimization problem in the first place.

There are three approaches to this problem. The first and simplest is to just generated a vector b of arbitrary length stochastically weighted by p. If b is long enough this will work fine, but there is no way to know this outside of fitting z and comparing the fit to real

world data. The second is to systematically generate permutations such that every mode and mode switch is represented in b at least some arbitrary number of times. With that prerequisite fulfilled b can then be extended to arbitrary length stochastically weighted by p. This method for generating b has the same downsides as the first. Except that if the system has a rare mode or mode switch that is critically slow it is guaranteed to be represented in the constraints. Lastly we can define a starting number of permutations $b(1)$ and every step add some arbitrary number of unique permutations resulting in $b(2)$. At the start of every step we solve $b(k) \leq E(k) \cdot z(k)$ for the minimal possible $z(k)$. Based on these intermittently solved $z(k)$ a stopping condition can be defined: if $\sum z(k)$ does not increase by some arbitrary percentage relative to $z(k-1)$ then the new elements in $b(k)$ did not constrain significantly more than $b(k-1)$ and we can solve the minimization problem with $b(k)$ and $E(k)$ otherwise add unique permutations to find $b(k+1)$ and repeat.

4-3-2 Linear Regression

If we don't require our estimate to fulfill $\mathbb{E}\left[\overline{\Gamma(N)}\right] \leq C \cdot z$ but are instead merely interested in $\mathbb{E}\left[\overline{\Gamma(N)}\right] \approx C \cdot z$ we can get a better estimate using a weighted least squares approach. With the previously defined vector P as the weights all we need to solve is this regression problem:

$$\min_{z} \sum_{i=1}^{\text{length}(b)} \left[P_i \cdot (b_i - E_{i,*} \cdot z)^2\right]$$

4-4 Methods To Reduce Calculation Time

Calculating all permutations of b and the corresponding rows of E is the calculation time bottleneck of solving the minimization problem. Therefore it is worth writing the function to calculate both in C such that it can be called in Octave. This is easily done as the only operations required are maximization and addition in nested for-loops with as the input one integer N and a three dimensional array containing A.

A second way to save calculation time is to calculate b in a more efficient way. Say we have a system where $L = 3$ and we want to fit on $N = 4$ that means three matrix max plus products per permutation. With $L^N = 3^4 = 81$ permutations that makes 243 matrix max plus products to calculate it conventionally. However if we first calculate all 9 permutations for $N = 2$ and then go through all permutations of those nine max plus multiplied by itself we get the same b_4 in the end:

$$\tilde{b}_2 = \begin{bmatrix} A^{(1)} \otimes A^{(1)} \\ A^{(1)} \otimes A^{(2)} \\ \vdots \\ A^{(3)} \otimes A^{(3)} \end{bmatrix} = \begin{bmatrix} A^{(1,1)} \\ A^{(1,2)} \\ \vdots \\ A^{(3,3)} \end{bmatrix}, \quad b_4 = \begin{bmatrix} \overline{A^{(1)} \otimes A^{(1)} \otimes A^{(1)} \otimes A^{(1)}} \\ \overline{A^{(1)} \otimes A^{(1)} \otimes A^{(1)} \otimes A^{(2)}} \\ \vdots \\ \overline{A^{(3)} \otimes A^{(3)} \otimes A^{(3)} \otimes A^{(3)}} \end{bmatrix} = \begin{bmatrix} \overline{A^{(1,1)} \otimes A^{(1,1)}} \\ \overline{A^{(1,1)} \otimes A^{(1,2)}} \\ \vdots \\ \overline{A^{(3,3)} \otimes A^{(3,3)}} \end{bmatrix}$$

Yet doing it this way it only takes $9 + 81 = 90$ matrix max plus products, a 63% reduction.

length(b_N)	# of $\otimes$ conventional	Optimal Permutation	# of $\otimes$ optimal	Relative Savings for $L = 3$
length(b_2) $= L^2$	$L^2 \cdot (2-1)$	$b_2 = \left[\tilde{b}_1, \tilde{b}_1\right]$	L^2	$\frac{9-9}{9} = 0\%$
length(b_3) $= L^3$	$L^3 \cdot (3-1)$	$b_3 = \left[\tilde{b}_2, \tilde{b}_1\right]$	$L^3 + L^2$	$\frac{54-36}{54} = 33.33\%$
length(b_4) $= L^4$	$L^4 \cdot (4-1)$	$b_4 = \left[\tilde{b}_2, \tilde{b}_2\right]$	$L^4 + L^2$	$\frac{243-90}{243} = 62.96\%$
length(b_5) $= L^5$	$L^5 \cdot (5-1)$	$b_5 = \left[\tilde{b}_3, \tilde{b}_2\right]$	$L^5 + L^3 + L^2$	$\frac{972-279}{972} = 71.30\%$
length(b_6) $= L^6$	$L^6 \cdot (6-1)$	$b_6 = \left[\tilde{b}_3, \tilde{b}_3\right]$	$L^6 + L^3 + L^2$	$\frac{3645-765}{3645} = 79.01\%$
length(b_7) $= L^7$	$L^7 \cdot (7-1)$	$b_7 = \left[\tilde{b}_4, \tilde{b}_3\right]$	$L^7 + L^4 + L^3 + L^2$	$\frac{13122-2304}{13122} = 82.44\%$
length(b_8) $= L^8$	$L^8 \cdot (8-1)$	$b_8 = \left[\tilde{b}_4, \tilde{b}_4\right]$	$L^8 + L^4 + L^2$	$\frac{45927-6651}{45927} = 85.52\%$

Table 4-1: Max-Plus matrix product savings as N increases for $L = 3$

The number of max-plus products required for the optimal method of calculating all permutations is as follows:

1. Start a counter $k = 1$ and set the first entry in a 1 dimensional array f_k of unspecified length $f_1 = N$.
2. If f_k is even go to step 3 if f_k is uneven go to step 4.
3. Set $f_{k+1} = \frac{f_k}{2}$ and increase the counter $k = k + 1$ go to step 5.
4. Set $a = \frac{1+f_k}{2}$ and set $b = f_k - a$. If a is even set $f_{k+1} = a$ and $f_{k+2} = b$ else set $f_{k+1} = b$ and $f_{k+2} = a$. increase the counter $k = k + 2$.
5. If $f_k \neq 2$ AND $f_{k-1} \neq 2$ go to step 2.
6. The number of max-plus products required to calculate b_N is calculated: $\sum_{i=1}^{k} L^{f_i}$

This means that the number of max plus products is limited: $\sum_{i=1}^{k} L^{f_i} \leq L^N + \mathcal{O}\left(L^{\frac{N+2}{2}}\right)$ such that we can write:

$$\begin{aligned} &\lim_{N\to\infty} \frac{L^N \cdot (N-1) - \left(L^N + \mathcal{O}\left(L^{\frac{N+2}{2}}\right)\right)}{L^N \cdot (N-1)} \\ &= \lim_{N\to\infty} \frac{L^N \cdot (N-2) - \mathcal{O}\left(L^{\frac{N+2}{2}}\right)}{L^N \cdot (N-1)} \\ &= \lim_{N\to\infty} \frac{N - 2 - \frac{\mathcal{O}\left(L^{\frac{N+2}{2}}\right)}{L^N}}{N-1} = 1 \end{aligned} \tag{4-10}$$

Which means that for higher N and L the relative savings can simply be described by $\frac{N-2}{N-1}$. This slightly underestimates the denominator and thus overestimates the savings but we take $N = 8$ we find $\frac{8-2}{8-1} = 85.74\% \approx 85.52\%$.

4-5 Memory Concerns

While this method is mostly constrained by the CPU it is nevertheless quite memory intensive aswell. Firstly to calculate $b_{N_{fit}}$ we need to calculate at least all full matrices of the intermediate b vectors that have to be multiplied to save on calculationtsteps as described in Table 4-1. In the worst case that means storing $\tilde{b}_{\frac{N_{fit}+1}{2}}$ and $\tilde{b}_{\frac{N_{fit}-1}{2}}$ as arrays of full $n \times n$ matrices. Storing these two three dimensional arrays as doubles takes: $\left(L^{\frac{N_{fit}+1}{2}} + L^{\frac{N_{fit}-1}{2}}\right) \cdot n^2 \cdot 2$ bytes. These arrays can be deleted once $b_{N_{fit}}$ is calculated which is only $2 \cdot L^{N_{fit}}$ bytes when stored as doubles as we only store the maximum value instead of the whole $n \times n$ matrix.

Alternatively the limiting factor could be the RAM required to solve the minimization problem as we have to store $b_{N_{fit}}$ and $E_{N_{fit}}$. Assuming we are fitting the second order extrapolatable z as described in (4-4) these two arrays when stored as doubles take $(1 + L + L^2) \cdot L^{N_{fit}}$ bytes.

If we define O_{fit} as the order for which we want to fit z we can find a generalized formula in bytes:

$$2 \cdot \max\left[L^{N_{fit}} \cdot \sum_{i=0}^{O_{fit}} L^i, n^2 \cdot \left((1 - \operatorname{mod}(N_{fit}, 2)) \cdot L^{\frac{N_{fit}}{2}} + \operatorname{mod}(N_{fit}, 2) \cdot \left(L^{\frac{N_{fit}+1}{2}} + L^{\frac{N_{fit}-1}{2}}\right)\right)\right] \tag{4-11}$$

Which just shows that in any case where calculating $b_{N_{fit}}$ is the limiting factor by a small margin choosing an even N_{fit} one integer above the current uneven value may resolve the issue.

4-6 Scheduling

When we solve z on a second or or higher order these higher order marginal delays are useful for scheduling the input of a system. Imagine for example our package sorter. What if instead of the order of the packages being entirely stochastic we implemented a buffer that could hold four packages. Every action step one package is chosen from the buffer to be processed and a new package flows into the buffer putting our count back at four. If we define R to be the set of packages currently in the buffer we could model this:

$$\begin{aligned}
r(k-1) &= \text{The last package to have entered the system} \\
r(k) &= \text{The next package to be insterted into the system from the buffer} \\
\tilde{r}(k+1) &= \text{The package second in line projected to be insterted from the buffer} \\
&\vdots \\
\tilde{r}(k+|R|-1) &= \text{The package } N^{th} \text{ in line projected to be insterted from the buffer} \\
\tilde{r}(k+|R|) &= \text{The next package to enter the buffer once } r(k) \text{ leaves}
\end{aligned}$$

Since all packages will need to go through the system eventually there is no use in considering the marginal delay of any individual package $d_{r(k)}$ for deciding the order of insertion.

Instead we should consider if marginal switching costs can be minimized. This is something that can be done especially well using this model of marginal delays and marginal switching costs. The S matrix approach is effectively a first order approach and by definition cannot consider marginal switching costs of a second order or beyond. And a fitted probability model approach is by definition agnostic to specific modes as it is effectively fitted to the probability distribution of modes. Or phrased differently a fitted probability distribution by definition does not have the information fidelity to inform us on what order of modes would be more optimal.

If we previously modeled the system to the second order that means we would consider $d_{r(k-1),r(k)}$ but if we fitted to the the third we should consider $d_{r(k-1),r(k)} + d_{r(k-2),r(k-1),r(k)}$ and so on. Secondly, with an inflow-outflow buffer should consider that only $r(k)$ is really going to be decided upon $\tilde{r}(k+1)$ and further out are merely projections as $\tilde{r}(k+1) \neq r(k+1)$. Therefore we define a vector w with $w_1 = 1 \geq w_2 \geq w_3 \geq ... \geq w_{|R|+1}$ such that we take into account this increasing uncertainty. If a system works by filling the buffer completely followed by emptying the entire buffer in some order, then these weights should be weighted $w_1 = 1 = w_2 = w_3 = ... = w_{|R|+1}$ needing only one calculation every full buffer or $|R|$ packages. For a second order controller the minimization function would work out as:

$$\begin{aligned} &\min_{r(k)...r(k+|R|)} \sum_{i=1}^{|R|} w_i \cdot d_{r(i-1),r(i)} + w_{|R|+1} \sum_{i=1}^{L} p_i \cdot d_{r(k+|R|-1),r(k+|R|)} \\ &\text{such that:} \\ &\{r(k), r(k+1), ..., r(k+|R|-1)\} = R \end{aligned} \tag{4-12}$$

We do not see an elegant way to convert this into a optimization problem. As such R probably needs to be searched by going through all possible permutations for every step. For systems with relatively small buffers and few modes this is hardly a problem. For systems with large buffers and many modes and relatively equally distributed odds this search space quickly explodes as it is all possible unique permutations of the packages currently in the buffer. In such cases a heuristic, such as a genetic algorithm, or simply randomly attempting 1000 permutations and picking the best one, may be able to find an acceptable local optimum sufficiently quickly.

4-6-1 Effect of Control

In order to asses if controlling systems this way is in any way effective we preformed a Monte Carlo Simulation. Systems were randomly generated with: $L \in \{2, 3, 4\}$, $n \in \{2, 3, 4, 5, 6\}$ and between 0% to 60% of elements equal to ε. Non-ε elements were calculated as the minimum of two independent random variables $X \in [0, 10]$ that are uniformly distributed. This way of generating elements was done so as to generate systems that are more likely to have higher marginal switching costs making the ordering of the modes more significant. For every system we also generate random elements in p such that $\sum_{i=1}^{L} p_i = 1$ and then a 25 item integer array $m \in \mathbb{Z}^{1\times 25}$ of modes with every element $m(i) \in \{1, 2, ..., L\}$ weighted according to p.

For every random system we then calculate a vector $y \in \mathbb{R}^{1\times 25}$ with $y(k) = \bigotimes_{i=1}^{k} A^{(m(i))}$ as the base not controlled case. Then for the 5 item buffer second order case we feed the scheduling

algorithm described in (4-12) $m(1)$ through $m(5)$ as a startup for simplicity sake we choose a uniform weighting $w_i = 1,\ \forall i$. During this first sort we cannot consider any $r(i-1)$ as the buffer was empty up to this point which means the first part of the optimization function is $\sum_{i=2}^{|R|} w_i \cdot d_{r(i-1),r(i)}$ uniquely for this first sort. We then take the first item from the initialized sorted buffer R and set it to $m_{2ndOrder5Buffer}(1)$ and append the buffer R back up to five items with $m(6)$. We once again sort that buffer according to (4-12) take the first item again and set it to $m_{2ndOrder5Buffer}(2)$ and append the buffer with $m(7)$. We continue this process until $m(25)$ has been added to the buffer at which point we sort it one last time now not considering the second part of the minimization function $w_{|R|+1} \sum_{i=1}^{L} p_i \cdot d_{r(k+|R|-1),r(k+|R|)}$ as there will be no $m(26)$ added. We then perform the same process but now with a buffer size of $|R| = 10$ so as to find $m_{2ndOrder10Buffer}$.

We apply nearly the same process to calculate the effects of third order control with some differences. Firstly, since a third order process considers not only the one package before the current one but two packages before the current one we need to feed the optimization function $r(i-1)$ as well as $r(i-2)$. Secondly, we have to rewrite (4-12) to actually be third order (4-13). Thirdly, this third order minimization function takes a third order vector d which needs to be fitted to the system before any scheduling can be done.

$$\begin{aligned} &\min_{r(k)\ldots r(k+|R|)} \sum_{i=1}^{|R|} w_i \cdot d_{r(i-1),r(i)} + w_{|R|+1} \sum_{i=1}^{L} p_i \cdot d_{r(k+|R|-1),r(k+|R|)} \\ &\text{such that:} \\ &\{r(k), r(k+1), ..., r(k+|R|-1)\} = R \end{aligned} \tag{4-13}$$

Then using all four controlled arrays $m_\star(k)$ we can calculate all $y_\star(k) = \bigotimes_{i=1}^{k} A^{(m_\star(i))}$ so as to be able to compare them to the no control base case $y(k)$. An example of one Monte Carlo simulation can be found in Figure 4-5.

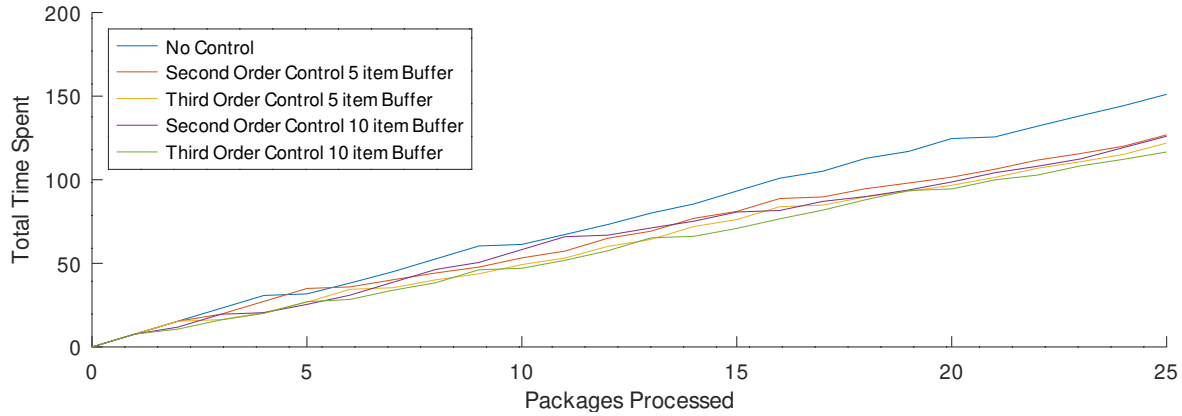

Figure 4-5: Example of a single Monte Carlo simulation

In this case control saves somewhere between 18% to 22% of sorting time and one would expect a second order small buffer approach is less effective than a large buffer third order approach. However in some rare cases this is not the case. Presumably this is due to the relatively low number of simulation steps or phrased differently low number of packages as

we only simulated up to $N = 25$ which is only 2.5 buffers worth in the case of the large buffer. Secondly, we only fitted the models to $N_{fit} = 7$ and fitting higher orders effectively requires higher N_{fit} for a quality fit as such the third order fits are presumably marginally less precise. Nevertheless, as can be seen in Figure 4-6 the five item second order scheduling function generally performs worst and the third order large buffer approach performs best.

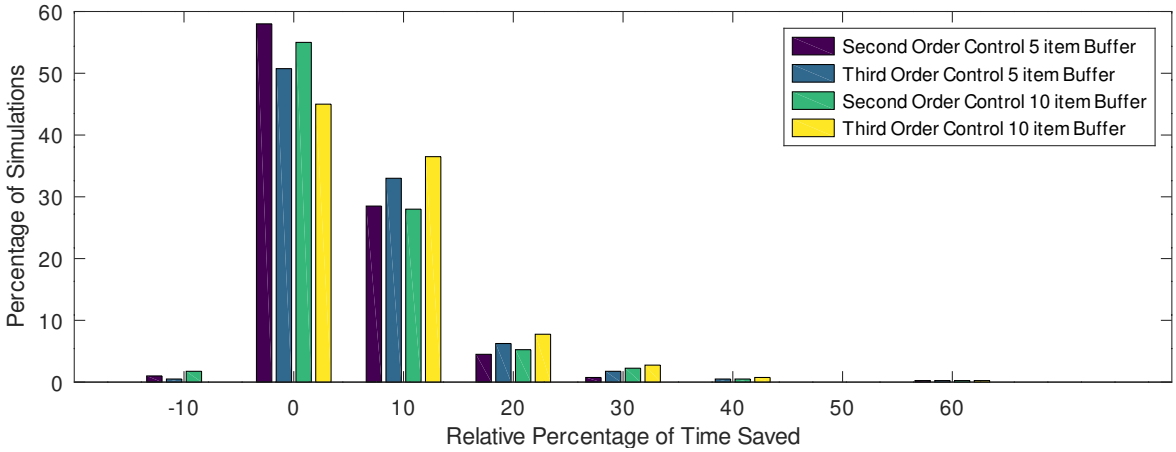

Figure 4-6: Histogram of the relative effect of scheduling

The histogram in Figure 4-6 was generated by simply calculating the relative savings in percent $s_\star = \left(1 - \frac{y(25)}{y_\star(25)}\right) \cdot 100$. If we define $\mathcal{M}$ to be the set of all simulations such that $|\mathcal{M}| = 250$ we can calculate the average relative time saves: $m_\star = \frac{1}{|\mathcal{M}|} \sum^{\mathcal{M}} s_\star$ $m_{2ndOrder5Buffer} = 4.6451$, $m_{3rdOrder5Buffer} = 6.0716$, $m_{2ndOrder10Buffer} = 5.5101$ and $m_{3rdOrder10Buffer} = 7.1426$. This again confirms that a higher order approach and larger buffer will probably perform best on $N_{fit} \to \infty$ and $N \to \infty$. But there are diminishing returns and the calculation load explodes superexponentially as you increase buffersize and the Order of the control function.

It would be useful if one could estimate whether and to what extend a system will be fit for scheduling without having to do simulations. One might think that the larger the relative switching costs relative to the unavoidable marginal cost per package as described in (4-14) would be a good predictor.

$$w_{2ndOrder} = \frac{\sum_{i=1,j=1}^{L} |d_{i,j}|}{L \cdot \sum_{i=1}^{L} |d_i|}, \quad w_{3rdOrder} = \frac{L \cdot \sum_{i=1,j=1}^{L} |d_{i,j}| + \sum_{i=1,j=1,k=1}^{L} |d_{i,j,k}|}{L^2 \cdot \sum_{i=1}^{L} |d_i|} \tag{4-14}$$

However if we scatter plot $w_{2ndOrder}$ against $m_{2ndOrder5Buffer}$ and do the same for the other three simulations we get Figure 4-7. And one can immediately see that there is no significant correlation there which is confirmed by the actual correlation coefficients that range from -0.04 to 0.07.

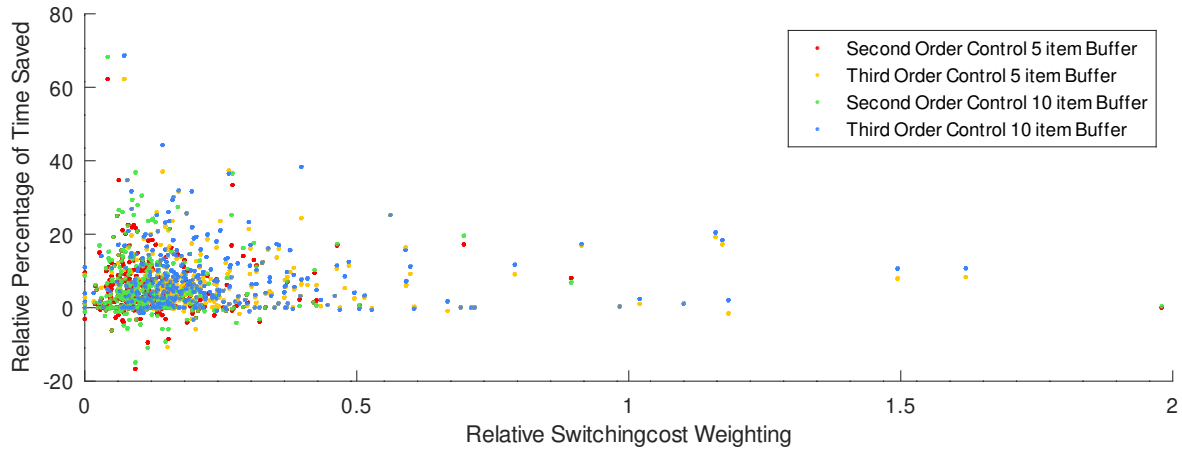

Figure 4-7: Relative Error for every fit on an N=8 simulation

However a second point to consider is that if all the switching cost options are equally slow meaning $d_{1,1} \approx d_{1,2} \approx d_{2,1}$ and so forth scheduling hardly makes a difference. So what we presumably really want is high variance in the switching costs and for them to still be weighted heavily relative to the inescapable marginal mode cost. In order to express this we can adjust (4-14) to also include variance as we do in (4-15).

$$
\begin{aligned}
v_{2ndOrder} &= \frac{\sum_{i=1,j=1}^{L^2} \left(d_{i,j} - \overline{d_{\star,\star}}\right)^2 \cdot \sum_{i=1,j=1}^{L} |d_{i,j}|}{L^3 \cdot \sum_{i=1}^{L} |d_i|} \\
v_{3rdOrder} &= \frac{L^2 \cdot \sum_{i=1,j=1}^{L^2} \left(d_{i,j} - \overline{d_{\star,\star}}\right)^2 \cdot \sum_{i=1,j=1}^{L} |d_{i,j}| + \sum_{i=1,j=1,k=1}^{L^2} \left(d_{i,j,k} - \overline{d_{\star,\star,\star}}\right)^2 \cdot \sum_{i=1,j=1,k=1}^{L} |d_{i,j,k}|}{L^5 \cdot \sum_{i=1}^{L} |d_i|}
\end{aligned}
\tag{4-15}
$$

If we then again scatter plot relative error against this new $v_{2ndOrder}$ and $v_{3rdOrder}$ we get Figure 4-8. In which we surprisingly find there is still no significant correlation at all as such we find no easy predictor for how effective control might be.

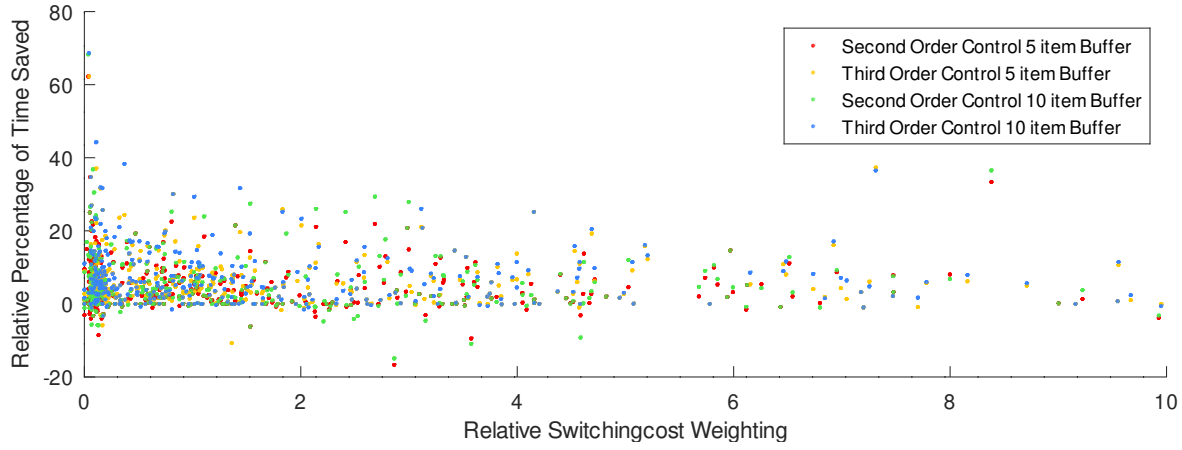

Figure 4-8: Relative Error for every fit on an N=8 simulation

Chapter 5

Estimating $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ using a Probability Distribution

Another way to describe these systems is to just fit an extreme value model and extrapolate on that model. This method does not give any guarantees such as that your estimation will always be larger or equal to $\mathbb{E}[\Gamma(N)]$. However all previously discussed models only estimate $\mathbb{E}[\Gamma(N)]$ which lies at in the middle of the probability distribution with 50% chance of overshoot or undershoot of any specific $\Gamma(N)$. If one wanted to know the maximum time that any process could take with a 95% certainty $\mathbb{E}[\Gamma(N)]$ is of little use. However when you have a fitted probability distribution these questions can be answered.

In order to find a probability density function for some N_{fit} we need to calculate a $b_{N_{fit}}$ which is probably most efficiently done using the method described in Section 4-4. Secondly, we need to know the odds for every element in this $b_{N_{fit}}$ this is described by:

$$P_{N_{fit}} = \begin{bmatrix} P(\ell_1 = 1, \ell_2 = 1, ..., \ell_{N_{fit}-1} = 1, \ell_{N_{fit}} = 1) \\ P(\ell_1 = 1, \ell_2 = 1, ..., \ell_{N_{fit}-1} = 1, \ell_{N_{fit}} = 2) \\ \vdots \\ P(\ell_1 = 1, \ell_2 = 1, ..., \ell_{N_{fit}-1} = 1, \ell_{N_{fit}} = L) \\ P(\ell_1 = 1, \ell_2 = 1, ..., \ell_{N_{fit}-1} = 2, \ell_{N_{fit}} = 1) \\ \vdots \\ P(\ell_1 = L, \ell_2 = L, ..., \ell_{N_{fit}-1} = L, \ell_{N_{fit}} = L) \end{bmatrix} = \begin{bmatrix} p_1 \cdot p_1 \cdot ... \cdot p_1 \cdot p_1 \\ p_1 \cdot p_1 \cdot ... \cdot p_1 \cdot p_2 \\ \vdots \\ p_1 \cdot p_1 \cdot ... \cdot p_1 \cdot p_L \\ p_1 \cdot p_1 \cdot ... \cdot p_2 \cdot p_1 \\ \vdots \\ p_L \cdot p_L \cdot ... \cdot p_L \cdot p_L \end{bmatrix}$$

Say we randomly generate a system such that we can calculate $b_{N_{fit}}$:

$$A^{(1)} \approx \begin{bmatrix} 6.29 & 7.07 & 1.82 \\ 1.42 & \varepsilon & 0.79 \\ 9.90 & 1.45 & \varepsilon \end{bmatrix}, A^{(2)} \approx \begin{bmatrix} 5.93 & 4.36 & 4.21 \\ \varepsilon & 3.31 & \varepsilon \\ 8.22 & 8.34 & 5.24 \end{bmatrix}, A^{(3)} \approx \begin{bmatrix} \varepsilon & 0.15 & \varepsilon \\ 0.97 & 6.51 & \varepsilon \\ 5.33 & 7.89 & 9.13 \end{bmatrix}, p \approx \begin{bmatrix} 0.41 \\ 0.39 \\ 0.20 \end{bmatrix}^T \tag{5-1}$$

Using this system we calculate out to $N_{fit} = 7$ such that we optain b_7 and P_7. with these two we can plot two histograms: once unweighted and once with every element of $\frac{b_7}{7}$ weighted by by its corresponding element in P_7. We scale by $\frac{1}{7}$ such that we again get to the marginal delay for each step rather than the total processing time. In most systems the difference between the weighted and unweighted histograms is relatively subtle, the height of the peak may differ by 10% and the location of the peak may differ by 15%. The same is the case for our example system as can be seen in Figure 5-1.

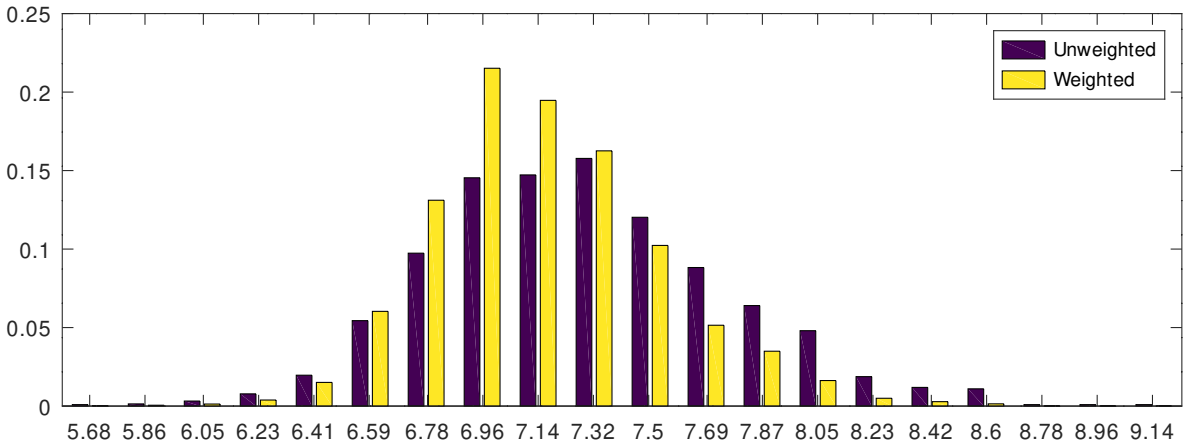

Figure 5-1: Histogram counting the occurrence of entries in b and then the occurrence of those same elements by weighted by odds

5-1 Inherent imprecision to extrapolating beyond N_{fit}

It should be noted that this method of calculating the marginal delay by definition risks underestimating switching costs. For example in the case of $N_{fit} = 7$ we consider 7 modes but therefore only 6 mode-switches. This is fine if you want to make a prediction for processing time on a number of packages exactly equal to N_{fit} and no packages will precede or follow the section you are attempting to predict. But if you wanted to make a prediction for some section of length N_{fit} in the middle of the processing sequence, packages will come after and have come before the section you are estimating. Alternatively if one wanted to predict $2 \cdot N_{fit}$ or $0.5 \cdot N_{fit}$ number of packages one would also underweight switching costs.

In the first case since we know the predicted section is in the middle of a queue we would either have to consider the switching cost from the package before our first considered package, or the switching costs beyond our last considered package. If we do not we underweight switching costs by a factor $1 - \frac{N_{fit}-1}{N_{fit}} = \frac{1}{N_{fit}}$.

In the second case if we estimate short of N_{fit} we overestimate switching costs and if we estimate beyon N_{fit} we underestimate them. Say $N_{fit} = 4$ that means we fitted 4 marginal mode delays but only 3 mode switches. If we were to estimate for $N = 2$ scaling the model gets us 2 marginal modes and 1.5 switches which is 0.5 too many. The other way around if we extrapolate to $N = 8$ we consider only 6 mode switches where we should consider 7.

And that is assuming there are only interactions on the second order, in many systems there will be interactions on the third order and beyond. Therefore if the marginal delays of

modes themselves do not dominate the total processing time, this method is highly unfit for calculating $\mathbb{E}[\Gamma(N)]$ especially when fitting at a low N_{fit} and extrapolating the resulting model to $N >> N_{fit}$.

5-2 Fitting the Generalised Extreme Value distribution

We use the Generalised Extreme Value (GEV) distribution to fit this data. This model has 3 special cases, the Fréchet distribution, the Gumbell distribution or the Weibull distribution. If the generalised fit consistently overlaps with those of any of these three special cases and its shape parameter ξ signals a special case we can assume it is safe to use this particular special case going forward.

We could either fit the Probability Density Function (PDF) of these models to the histogram in Figure 5-1 of course we would first have to scale this histogram such that the sumtotal surface area of these bars equals 1. Alternatively we could fit the Cumulative Density Function (CDF) in (5-2) to a scatterplot of every element in $b_{N_{fit}}$. The latter seems like the better method as creating a histogram by definition loses information when grouping the datapoints, which can only result in a lower quality fit. Secondly the PDF formula's are more complex to calculate for every one of the models, so fitting on the CDF should save us calculation time as well.

$$
\begin{aligned}
&\mathrm{CDF}_{Generalized}(x,\xi,\sigma,\mu) = \begin{cases} e^{-\max\left[0,1+\xi\cdot\left(\frac{x-\mu}{\sigma}\right)\right]^{-\frac{1}{\xi}}}, & \xi \neq 0 \\ e^{-e^{-\frac{x-\mu}{\sigma}}} & \xi = 0 \end{cases} \\
&\textbf{with: } \xi \in (-\infty,\infty) \textbf{ shape, } \sigma \in (0,\infty) \textbf{ scale, } \mu \in (-\infty,\infty) \textbf{ minimum} \\
&\mathrm{CDF}_{Fréchet}(x,\alpha,s,m) = e^{-\left(\frac{x-m}{s}\right)^{-\alpha}}, \quad \textbf{Special case for: } \xi \gg 0 \\
&\textbf{with: } \alpha \in (0,\infty) \textbf{ shape, } s \in (0,\infty) \textbf{ scale, } m \in (-\infty,\infty) \textbf{ minimum} \\
&\mathrm{CDF}_{Gumbel}(x,\beta,\mu) = e^{-e^{-\frac{x-\mu}{\beta}}}, \quad \textbf{Special case for: } \xi = 0 \\
&\textbf{with: } \beta \in (0,\infty) \textbf{ scale, } \mu \in (-\infty,\infty) \textbf{ location peak of density function} \\
&\mathrm{CDF}_{Weibull}(x,k,\lambda,m) = \begin{cases} 1-e^{-\left(\frac{x-m}{\lambda}\right)^{k}}, & x-m \geq 0 \\ 0, & x-m < 0 \end{cases}, \quad \textbf{Special case for: } \xi \ll 0 \\
&\textbf{with: } k \in (0,\infty) \textbf{ shape, } \lambda \in (0,\infty) \textbf{ scale, } m \in (-\infty,\infty) \textbf{ minimum}
\end{aligned}
\tag{5-2}
$$

In order to create the scatterplot we can fit the CDF's to we first need to sort $b_{N_{fit}} \rightarrow \hat{b}_{N_{fit}}$ such that $[\hat{b}_{N_{fit}}]_1 \leq [\hat{b}_{N_{fit}}]_2 \leq ... \leq [\hat{b}_{N_{fit}}]_{L^{N_{fit}}-1} \leq [\hat{b}_{N_{fit}}]_{L^{N_{fit}}}$. Next we also reorder $P_{N_{fit}} \rightarrow \hat{P}_{N_{fit}}$ to the same order such that we preserve match between the elements of b and P: $P\left(x = \left[\hat{b}_{N_{fit}}\right]_i\right) = \left[\hat{P}_{N_{fit}}\right]_i$. Then we need to sum $\left[\tilde{\hat{P}}_{N_{fit}}\right]_i = \sum_{j=1}^{i}\left[\hat{P}_{N_{fit}}\right]_i$ such that: $P\left(x \leq \left[\hat{b}_{N_{fit}}\right]_i\right) = \left[\tilde{\hat{P}}_{N_{fit}}\right]_i$. If we then create a scatterplot with the coordinates of every dot at $\left(\left[\hat{b}_{N_{fit}}\right]_i, \left[\tilde{\hat{P}}_{N_{fit}}\right]_i\right)$ using $\frac{b_7}{7}$ and P_7 calculated from the system described in (5-1) this results in the blue dots of Figure 5-2.

We can then set the minimum m and in some cases the location μ parameter before fitting. As $m = \left[\hat{b}_{N_{fit}}\right]_1$ and μ such that it corresponds to the steepest section of the scatterplot. This means we have to pick μ such that:

$$\begin{aligned} &\mu = \left[\hat{b}_{N_{fit}}\right]_i \\ &\text{such that:} \\ &\max_{i\in[s+1,L^{N_{fit}}-s]} \left[\left[\hat{b}_{N_{fit}}\right]_{i+s} - \left[\hat{b}_{N_{fit}}\right]_{i-s}\right] \end{aligned} \tag{5-3}$$

We can arbitrarily pick an integer $s \in [1, \infty)$ such that we consider a wider difference between dots to minimize the effect of noise in the scatterplot. This works with fairly low s for reasonably smooth systems such as the one in Figure 5-2 but in some cases systems are highly non-smooth such as the one to be discussed in Figure 5-7. These systems are generally not wel described by any CDF but if one still wanted to fit such a system it is better to fit μ instead of predefining it.

We then fit the scale and shape parameters q_1 and q_2 and in some cases the location parameter q_3 such that we minimize the squared error:

$$e_{type} = \min_{q_1\in\{\xi,\alpha,-,k\},q_2\in\{\sigma,s,\beta,\lambda\},q_3\in\{\mu,m,\mu,m\}} \left[\sum_{i=1}^{L^{N_{fit}}} \frac{\left(\left[\tilde{\hat{P}}_{N_{fit}}\right]_i - \text{CDF}_{type}\left(\left[\hat{b}_{N_{fit}}\right]_i, q_1, q_2, q_3\right)\right)^2}{L^{N_{fit}}}\right] \tag{5-4}$$

For the system described in (5-1) the fitted CDF's are plotted alongside the raw scatter data in Figure 5-2.

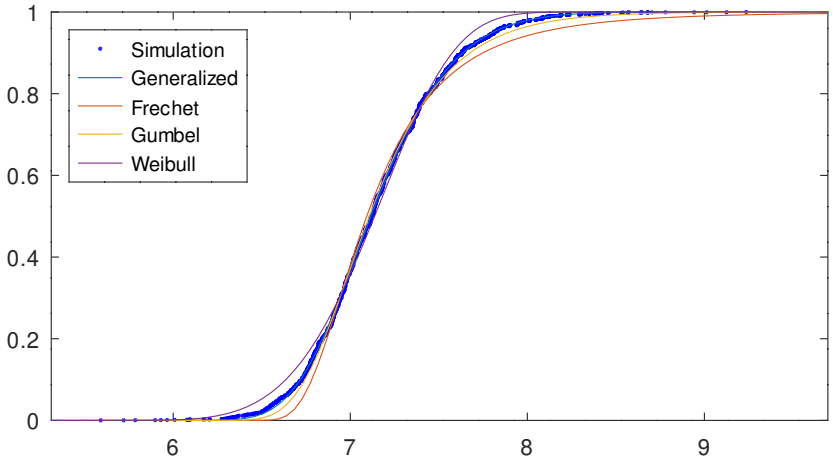

Figure 5-2: Fitting cumulative extreme value distributions to the simulation

In the case of this specific system the squared errors are $e_{Generalized} = 2.2189 \cdot 10^{-5}, e_{Fréchet} = 7.6790 \cdot 10^{-4}, e_{Gumbel} = 1.3813 \cdot 10^{-4}, e_{Weibull} = 5.2665 \cdot 10^{-4}$ with $\xi = -0.1207$. But we can't say much based on 1 system thus we did a Monte Carlo simulation over 100 randomly generated systems for which the mean errors worked out as: $e_{Generalized} = 2.6967 \cdot 10^{-4}, e_{Fréchet} =$

$1.2487 \cdot 10^{-2}, e_{Gumbel} = 3.4464 \cdot 10^{-3}, e_{Weibull} = 1.1624 \cdot 10^{-2}$. These average errors would suggest no special case with even the gumbel distribution scoring almost an order of magnitude worse than the generalized distribution. This a spread on mean errors would lead one to think ξ is uniformly distributed with perhaps a slightly higher density around $\xi = 0$. But if we plot a histogram for all ξ found in Figure 5-3 we see it clusters around $\mathbb{E}[\xi] \approx -0.3$. Which would suggest the Weibull distribution should fit best in contrast to the high mean error.

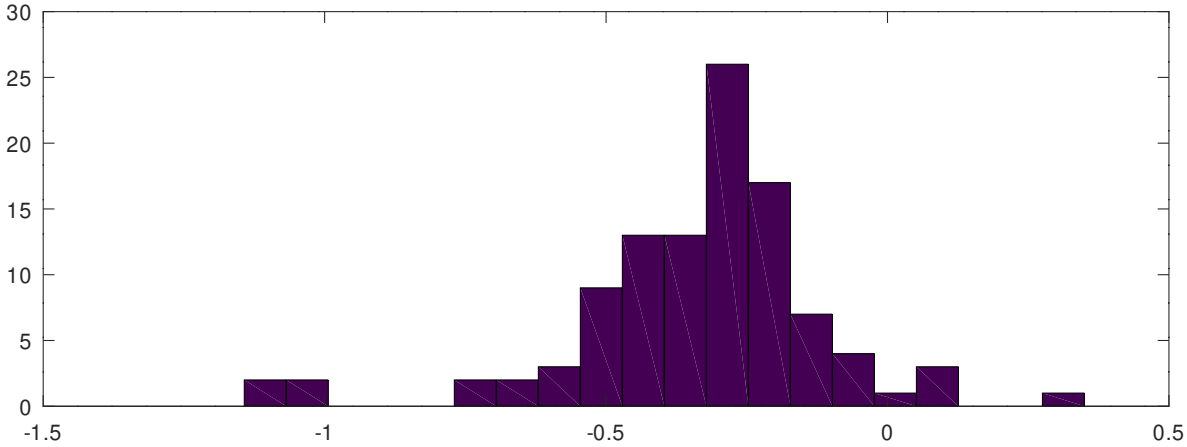

Figure 5-3: Histogram of ξ based on Monte Carlo simulations

To resolve this apparent paradox we plot every system's fitted ξ out against the error for every probability model as is done in Figure 5-4. At a first glance it looks as though the Weibull distribution and to a lesser extend the other two are performing much better than the average error would suggest. But at closer inspection we find a large difference between the mean and median error due to extreme outlier errors of orders larger than $\mathcal{O}(10^{-2})$. If we look at these outlier systems in detail we find they are all non-smooth like the system in Figure 5-7 with some $p_m \gg \{p_1, ..., p_{m-1}, p_{m+1}, .., p_L\}$.

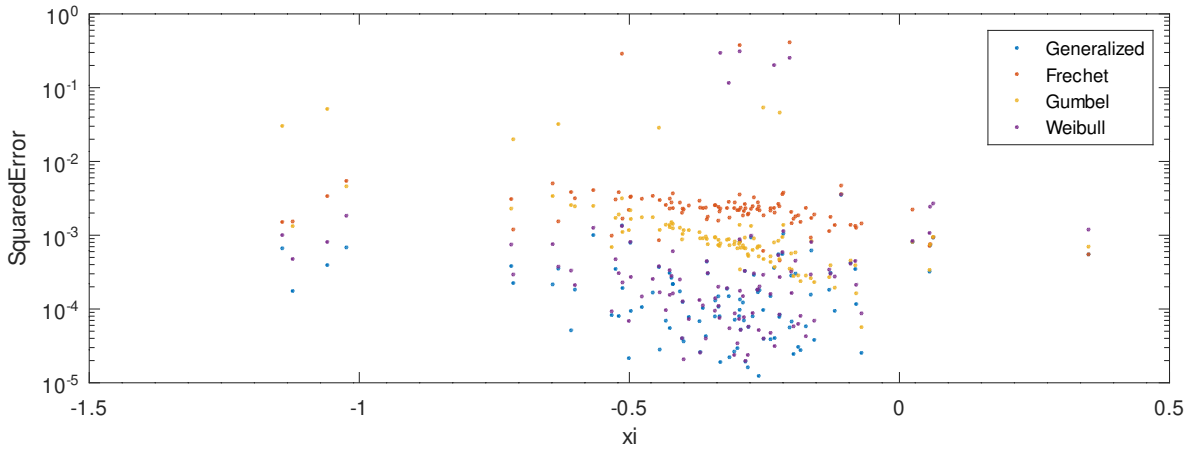

Figure 5-4: Scatterplot of ξ vs e_{type} based on Monte Carlo simulations

If we then exclude these problematic systems and calculate the mean errors over the remaining

92 smooth simulations we find that the Weibull distribution performs closest to the generalized distribution as can be seen in Table 5-1. Secondly, we now find both the Frechet and Gumbel distribution correlate negatively with ξ which is expected as for almost all datapoints $\xi < 0$. The Generalized an Weibull distribution do not correlate with ξ significantly. We conclude that one should fit to the generalized distribution as it still clearly outperforms the Weibull distribution on the domain containing 95% of the fitted systems $\xi \in (-0.75, -0.1)$.

Type	Mean Error	e_{type} Correlation ξ	e_{type} Correlation $e_{Generalized}$
Generalized	$2.6186 \cdot 10^{-4}$	$1.5716 \cdot 10^{-2}$	1
Frechet	$2.3488 \cdot 10^{-3}$	$-5.5418 \cdot 10^{-1}$	$3.1654 \cdot 10^{-1}$
Gumbel	$1.0080 \cdot 10^{-3}$	$-6.3560 \cdot 10^{-1}$	$4.7524 \cdot 10^{-1}$
Weibul	$3.8469 \cdot 10^{-4}$	$1.6181 \cdot 10^{-1}$	$8.6544 \cdot 10^{-1}$

Table 5-1: Mean Error and Correlations excluding outlier systems

Using the distribution parameters we can also create the probability density distributions described in (5-5) for every system and compare it to its weighted histogram.

$$
\begin{aligned}
t(x,\xi,\sigma,\mu) &= \begin{cases} \max\left[0, 1+\xi\cdot\left(\frac{x-\mu}{\sigma}\right)\right]^{-\frac{1}{\xi}} & ,\xi \neq 0 \\ e^{-\frac{x-\mu}{\sigma}} & ,\xi = 0 \end{cases} \\
\mathrm{PDF}_{Generalized}(x,\xi,\sigma,\mu) &= \frac{t(x,\xi,\sigma,\mu)^{\xi+1}\cdot e^{-t(x,\xi,\sigma,\mu)}}{\sigma} \\
\mathrm{PDF}_{Fréchet}(x,\alpha,s,m) &= \frac{\alpha}{s}\cdot\left(\frac{x-m}{s}\right)^{-1-a}\cdot e^{-\left(\frac{x-m}{s}\right)^{-\alpha}} \\
\mathrm{PDF}_{Gumbel}(x,\beta,\mu) &= \frac{e^{-\frac{x-\mu}{\beta}-e^{\frac{x-\mu}{\beta}}}}{\beta} \\
\mathrm{PDF}_{Weibull}(x,k,\lambda,m) &= \begin{cases} \frac{k}{\lambda}\cdot\left(\frac{x-m}{\lambda}\right)^{k-1}\cdot e^{-\left(\frac{x-m}{\lambda}\right)^{k}} & ,x-m \geq 0 \\ 0 & ,x-m<0 \end{cases}
\end{aligned}
\tag{5-5}
$$

If we do this for the system defined in (5-1) we get Figure 5-5.

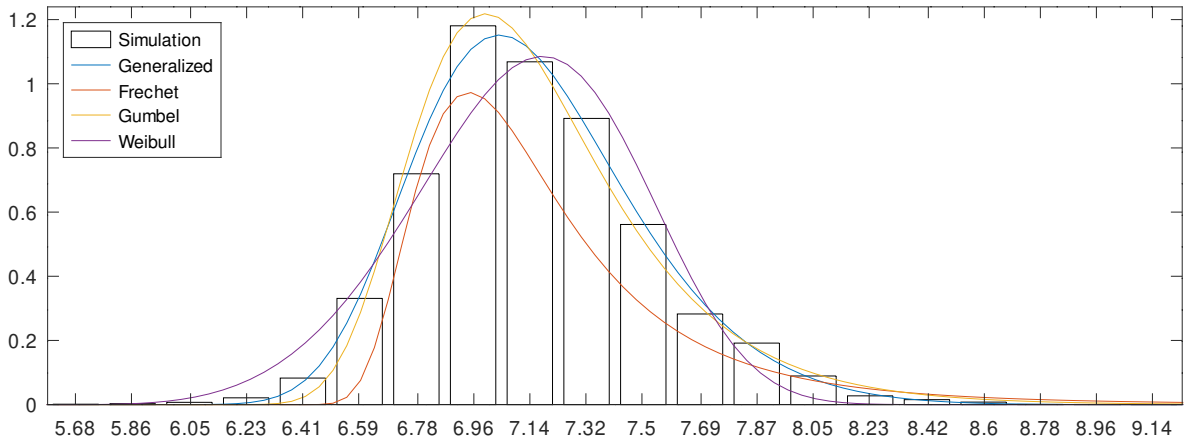

Figure 5-5: Fitting cumulative extreme value density functions to the simulation

5-3 Estimating $\mathbb{E}\left[\overline{\Gamma(N)}\right]$

Since we fit on $\frac{b_{N_{fit}}}{N_{fit}}$ we can just extrapolate the mean of the CDF's as $P(x \leq \frac{\mathbb{E}\left[\overline{\Gamma(N)}\right]}{N}) = \text{CDF}(x \leq \frac{\mathbb{E}\left[\overline{\Gamma(N)}\right]}{N}) \approx 0.5$. This 50^{th} percentile can just be calculated by taking the mean of the cumulative density function:

$$expectation_{Generalized}(x,\xi,\sigma,\mu) = \begin{cases} \mu + \frac{\sigma\cdot\left(\int_0^\infty \left(x^{\xi}\cdot e^{x}\right)dx - 1\right)}{\xi} & \xi \neq 0, \xi < 1 \\ \mu + \sigma \cdot \int_1^\infty \left(-\frac{1}{x} + \frac{1}{\lfloor x \rfloor}\right) & \xi = 0 \\ \infty & \xi \geq 1 \end{cases} \tag{5-6}$$

$$expectation_{Weibull}(x,k,\lambda,m) = \lambda \cdot \int_0^\infty \left(x^{\frac{1}{k}} \cdot e^{x}\right) dx$$

As this involves some symbolic math which can be quite slow we used a different method for approximating $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ during Monte Carlo simulations.

5-4 A note on non-smooth systems

Up to this point we have noted that some systems created outlier errors and as such were disregarded with the note that these systems should not be modelled using probability distributions. These systems are generally recognizable by two features. Firstly they have a highly uneven distribution of odds for modes to arise with generally one or two modes having more than 70% chance. Secondly this one or few modes have a marginal delay or switching cost that is either significantly higher or lower than that of the other low probability modes. This has as the result that he distribution function starts to look more like a staircase function and in the worst possible case like a step function. We can already somewhat achieve this effect by taking the system described in (5-1) and changing p such that $p = [0.19, 0.01.0.9]$. The unweighted Histogram in Figure 5-6 stays the same as the one in Figure 5-1 but when

weighting by odds you can already see how none of the PDF's in (5-5) could ever take on that shape.

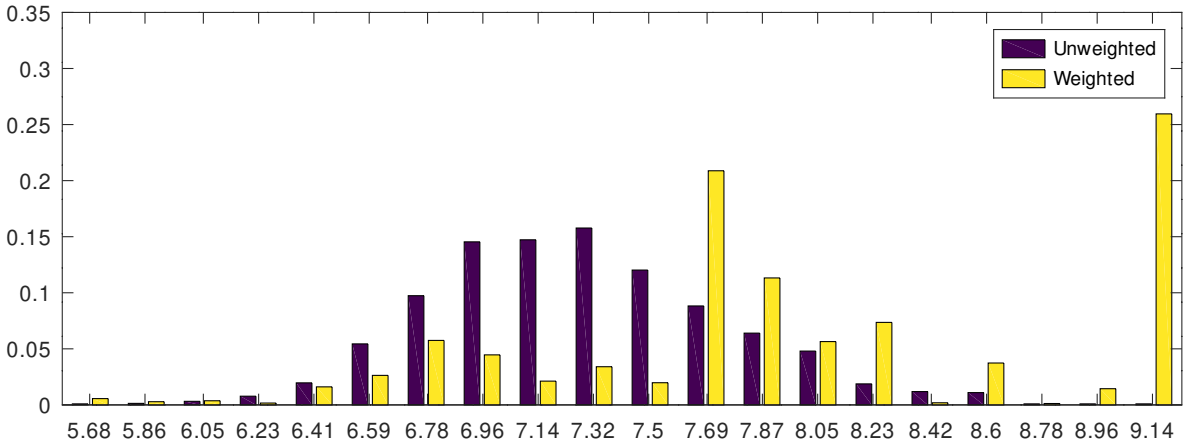

Figure 5-6: Histogram counting the occurrence of entries in B and then the occurrence of those same elements by weighted by odds

If we look at Figure 5-7 we can see the staircase like shape that is typical for the distribution scatterplot of these systems. And we can see the fitted distributions fit poorly $e_{Generalized} = 1.8786 \cdot 10^{-3}, e_{Frechet} = 4.0674 \cdot 10^{-3}, e_{Gumbal} = 2.7005 \cdot 10^{-3}, e_{Weibull} = 2.1696 \cdot 10^{-3}$.

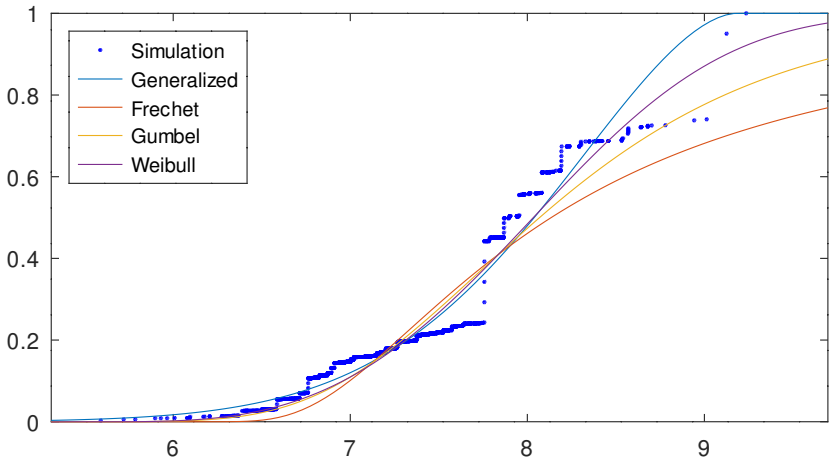

Figure 5-7: Fitting cumulative extreme value distributions to the simulation

5-5 Calculation time and Memory Concerns

Since we fit based on the same vector $b_{N_{fit}}$ that we use in Chapter 4 most memory and calculation time considerations can be applied here as well. Firstly if we are going through all permutations we can calculate $b_{N_{fit}}$ according to the algorithm described in Section 4-4 to reduce the number of max plus matrix products required, significantly reducing calculation time. Secondly as we want to fit with the highest possible N_{fit} so as to reduce the under-weighting of switching costs on $\lim_{N\to\infty} \mathbb{E}\left[\overline{\Gamma(N)}\right]$ we can generate an incomplete $b_{N_{fit}}$ as

discussed in Section 4-3-1. All we need to do is multiply all elements of $P_{N_{fit}}$ with some scalar g such that $g \cdot \sum P_{N_{fit}} = 1$ such that $\left[\tilde{\hat{P}}_{N_{fit}}\right]_{end} = 1$ as this is required for fitting a CDF. And lastly this method runs into one of the two memory bottlenecks mentioned in Section 4-5 we do not need to $E_{N_{fit}}$ so the memory bottleneck in bytes is calculated slightly differently in (5-7) than in (4-11):

$$2 \cdot \max\left[L^{N_{fit}}, n^2 \cdot \left((1 - \text{mod}(N_{fit}, 2)) \cdot L^{\frac{N_{fit}}{2}} + \text{mod}(N_{fit}, 2) \cdot \left(L^{\frac{N_{fit}+1}{2}} + L^{\frac{N_{fit}-1}{2}}\right)\right)\right] \quad (5\text{-}7)$$

One point specific to this method is fitting the distribution parameters. Since these are fitted using a non-linear optimization algorithm like Sequential Quadratic Programming (SQP) we want to minimize the number of iterations required to get an adequate fit. The first way to do this is to predefine m and μ as discussed in Section 5-2. If the system is non-smooth and μ can't be easily defined in advance then we can at least set the boundary conditions in the optimization options such that $\max\left[b_{N_{fit}}\right] \leq \mu \leq \max\left[b_{N_{fit}}\right]$ and set the seed for μ at $0.5 \cdot \left(\min\left[b_{N_{fit}}\right] + \max\left[b_{N_{fit}}\right]\right)$. Secondly, if you already have a good idea of what the other parameters should be one might consider limiting the search space and setting some seeding for these parameters. For example, if we were to do another Monte Carlo simulation like the one required for Figure 5-4 we might set the boundary conditions $-1.5 \leq \xi \leq 0.5$ and seeded at $\xi = -0.3$.

Chapter 6

Comparison of methods

To estimate the precision of different prediction methods we ran Monte Carlo simulations using randomly generated systems. None to 60 percent of elements in every mode could be set to ϵ the values of all non-ϵ elements of the matrices range between 0 and 10. The size of the systems range $n = 2$ and $n = 6$ with this limit mostly set to limit calculation time so as to allow for more simulations. The number of modes range $L = 2$ through $L = 5$ for mostly the same reason. The fact that such relatively simple systems were required to allow for many Monte Carlo iterations, already indicates the calculation time problem some methods run into when applied to larger systems. Nevertheless these results should extrapolate fairly well to larger systems with more modes except for the fact that higher L may require higher N_{fit} in some methods for the same quality prediction.

The first three items in the legend labeled "Action Counting1" through "Action Counting3" are approximations using the method described in Section 4-1-1 fitted to the first through third order respectively. The third order fit only begins at $N_{fit} \geq 3$ instead of $N_{fit} \geq 2$ as the N_{fit} always needs to be greater or equal to the order at which this method is fitted. We use the previously defined equation (4-7) to calculate the error for every Monte-Carlo simulation at every N_{fit} for all three orders of action counting predictions.

Secondly, as every method using the S matrices produce the same prediction, as they are just different methods for fitting the same S matrix, and the linear approach is fastest for small systems we only applied this method in the Monte-Carlo simulations. As the S matrix based approach is not fitted based on any N_{fit} the dots and dashed line labeled "Linear" have the exact same total error for every N_{fit}. We calculate this error by just taking the simulated total processing time substracted by the estimated total processing time: $Error = \mathbb{E}[\Gamma(8)] - P_8^T \cdot b_8$.

Lastly as the generalized extreme value distribution still significantly outperformed the Weibull distribution we only included a generalized distribution fit extrapolation. For which the error was calculated: $Error_{N_{fit}} = \mathbb{E}[\Gamma(8)] - P_8^T \cdot p_8$ with $\mathbb{E}[\Gamma(N)]$ as defined in (5-6).

For the These separate Monte-carlo simulations are represented by the dots in Figure 6-1. The dots do not fully align with the N_{fit} ticks on the x-axis so as to prevent different methods

from overlapping making the figure more readable as only integer N_{fit} are possible. The dashed line represents the mean of all simulations using that method at that particular N_{fit}.

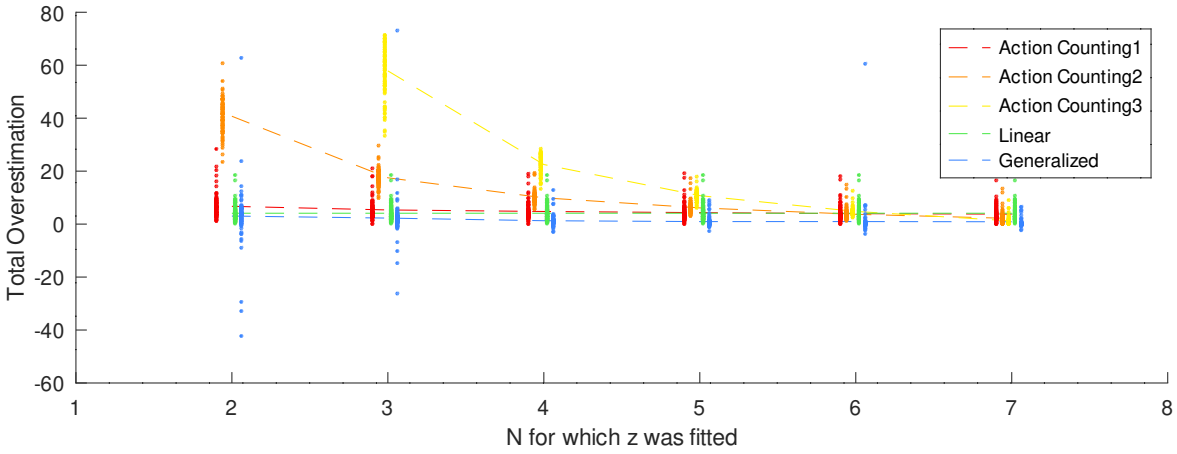

Figure 6-1: Histogram counting the occurrence of entries in b and then the occurrence of those same elements by weighted by odds

We can again use the previously defined equation (4-8) to calculate the relative error for the action counting approaches in Figure 6-2. For the S matrix approach and generalized extreme value distribution extrapolation we use:

$$RelativeError_{Nfit} = \frac{\left|\mathbb{E}\left[\Gamma(8)\right] - P_8^T \cdot b_8\right|}{P_8^T \cdot b_8} \cdot 100$$

This results in Figure 6-2 where we can mostly see what would be expected. Higher order approximations using the action counting method start out performing worse but for $N_{fit} \geq 6$ the fits of second and third order begin to outperform. The first order fit performs worse than an S matrix based approach for $N_{fit} \leq 6$ but begins to outperform from that point forward. Lastly, the generalized extreme value distribution outperforms all alternatives at least up to $N_{fit} \leq 7$ with the lowest variance on its relative error as well.

It must be noted however, that the parameters for generating random systems for this Monte-Carlo simulation were set to minimize the odds of generating non-smooth systems. While the other two methods do not have the same problems dealing with such systems. If we were to generate this error data including non-smooth systems there would be extreme outlier errors in the generalized extreme value distribution fit that would not get better at higher N_{fit} as discussed in Section 5-4. The action counting approach can run into a similar problem as described in Section 4-2 but this is rare and would affect the average less.

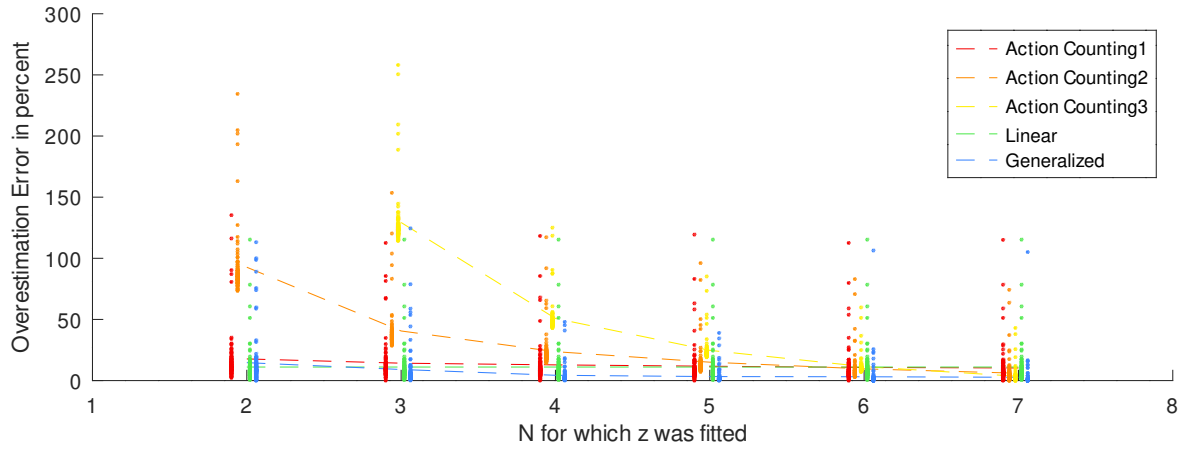

Figure 6-2: Histogram counting the occurrence of entries in b and then the occurrence of those same elements by weighted by odds

6-1 Correlation

The first order action counting approach and S matrix approach are very similar as they both effectively attempt to converge on the marginal delay of every mode. Which means that one would generally expect:

$$d_m \approx \overline{S^{\otimes} - A^{(\ell_m)} \otimes S} \quad , \forall m \in [1, L]$$

What makes it more interesting is that as N_{fit} increases, the correlation between the two becomes asymptotically less steep as can be seen in Figure 6-3, The slope first drops below one at $N_{fit} = 7$ the exact point at which the first order action counting method begins to outperform the S matrix based approach. Checking this correlation may be a quicker way to determine which of the two has better predictive capabilities than running a Monte-Carlo simulation to test which has the lower relative error.

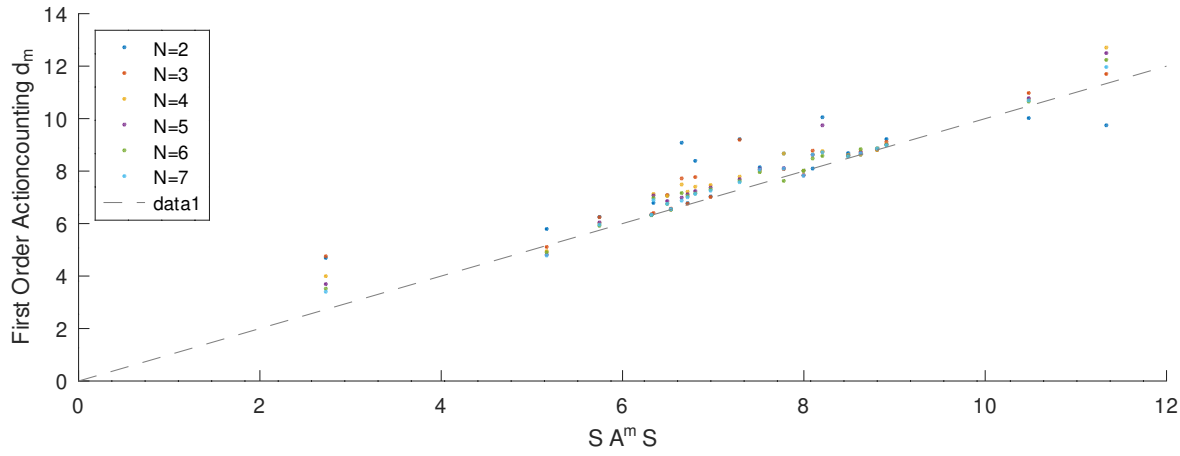

Figure 6-3: Histogram counting the occurrence of entries in b and then the occurrence of those same elements by weighted by odds

6-2 Applications

For large systems the S matrix approach is optimal. Especially so if it is required that the prediction is always greater than $\mathbb{E}[\Gamma(N)]$. At some very large L the better memory efficiency of the subgradient method described in (3-15) will allow it to fit faster than the linear approach. In all other cases the linear approach is faster to fit.

Simpler systems with less modes can be fitted using an action counting approach. The optimal fit is to pick the highest order fit that the chosen N_{fit} allows for. This has to be done as there is a trade off where higher order fits have a worse initial error but approach 0 more rapidly.

Alternatively using the methods described in Section 4-3 will mean the prediction may not always be larger or equal to $\mathbb{E}\left[\overline{\Gamma(N)}\right]$. But, implementing them will drop the relative error and they are therefore worthwhile if the specific relation is not relevant.

Lastly if the goal is to schedule the input of the system as discussed in Section 4-6 the only way to do so is using the action counting approach. Either fitted just as described in Section 4-1-1 or also including the methods described in Section 4-3. One should keep in mind though that the buffer always needs to be larger or equal to the order of the fit used for scheduling, and that at least a second order fit must be used.

Lastly, in any usecase where a prediction of some other percentile than the mean is wanted a generalized extreme value distribution can be fitted. As this method by definition cannot guarantee that its 50 percentile prediction is greater or equal to $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ fitting at a higher N_{fit} with an incomplete $b_{N_{fit}}$ as discussed in Section 4-3-1 will most likely result in a better fit.

Chapter 7

Conclusions and Recommendations

Coming around to the final chapter of this thesis we can look back at the research questions posed in the introduction. The first was wether we could predict the behaviour of a Switching Max-Plus-Linear (SMPL) system out into the future with an acceptable error:

Can we approximate the mean total processing time over N iterations for a stochastic SMPL system? At the very least the difficulty of calculating this approximation should scale sub-exponentially with increasing N while being at most 10% off the actual mean processing time.

And the second was whether we could possible invert any of these models and use them as a controller to reduce total processing time.

Can we exert some limited control over a stochastic SMPL system by using buffered modes in some way that is faster than simulating all possible permutations of inputting the buffer?

7-1 Conclusions

As Figure 6-2 shows the first research question can be answered yes for every method discussed in this Thesis as long as it is properly fitted. Which leads to the question: which method should be applied under what circumstances?

The simplest method to apply is fitting the S matrix approach using linear programming as discussed in Section 3-2. This method is very fast as solving a linear programming problem is a computationally easy when compared to the other methods. The sub-gradient based approach to fitting the S matrix discussed in Section 3-1 is only useful in the rare cases where a system has so many modes that the linear approach will not work. The only advantage of that approach is that it allows one to formulate the optimization function much more compactly, in all other ways it is more computationally intensive while still getting the same result as the linear approach at best.

In order to fit the other two methods we need to simulate all permutations of the system for some limited N_{fit} so as to generate $B_{N_{fit}}$ and $P_{N_{fit}}$. Doing this quickly becomes infeasible

for higher N_{fit} as the calculation steps to find $B_{N_{fit}}$ scales exponentially with N_{fit}. There are some methods to reduce the number of calculation steps as described in Section 4-4, these should be applied when possible.

If the best possible N_{fit} is limited due to little computational power or the system having many possible modes fitting a Generalised Extreme Value (GEV) distribution is best. If an upperbound for $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ is wanted the approximation found using a GEV distribution is unfit as it is just an approximation and not an upperbound. Secondly, if the system is non-smooth as described in Section 5-4 the fitted distribution will be unreliable.

For a sufficiently large N_{fit} it is possible to fit a sufficiently high order marginal cost model that can outperform a fitted GEV distribution. When fitting this model there is a trade off between N_{fit} and model order that needs to be tuned for the best fit. With the added benefit that the marginal cost model can be fitted to guarantee an upperbound on $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ rather than only estimating it. Secondly, having this model gives insights like what order of modes is more or less efficient which can be used for controlling the system inputs. Lastly, if the order of modes is not statistically independent this can be accounted for explicitly when extrapolating using this model.

As shown in Section 4-6 scheduling mode changes by exerting limited control on the inputs can be done. But, to answer the second research question we also need that limited control to decrease total processing time. Using at least an inverted second order marginal cost model it is possible to reduce total processing time in at least some systems as can be seen in Figure 4-6. In order to maximize the effect of control it is needed to first fit the highest order marginal cost model at the best N_{fit} possible and then use it to control the largest manageable buffer. Both are limited by the ability to solve the optimization problems within an acceptable time window.

7-2 Recommendations

In this section we will go through some recommendations for future research that might make the results found in this thesis more complete and applicable.

7-2-1 Better Buffer sorting Algorithm

In this thesis we set out to have some limited control over the mode switching process using an inverted marginal cost model as described in Section 4-6. In order to find the optimal feeding order for this buffer we just search the entire search space of possible feeding orders. This is perfectly manageable in for the purposes of this thesis as the worst case search space in those monte carlo simulations would be something like $\frac{10!}{2!2!3!3!} = 25200$ which would be quite rare to happen and even then was perfectly manageable. However in systems with many modes applying large buffers could in the worst case mean the search space scales factorially to buffer size.

These potentially large buffers probably can't reasonably be searched in their entirety as such it would be useful to find some heuristic to find acceptable buffers quickly. One way to attempt this might be to format the search space as a decision tree rather than a list of all

permutations, and abandon branches that do not seem promising when walking out from the stem to the leaves. There are probably many more ways to search the possible buffer feeding order and it might be useful to see which perform best.

7-2-2 Stochastic elements in A

In this thesis we have always considered the delays defined in the A matrices to be deterministic. In real applications this is almost never the case, as such it would be interesting to see if some of the prediction methods described in this thesis could be adapted to consider normally distributed elements in A.

7-2-3 Relationship between S matrix and prediction accuracy

When testing the code for doing the monte carlo simulations of the different S matrix based methods it seemed like there was a relationship between the elements on the diagonal of the S matrix and the accuracy of the predictions. Further research on whether there is such a relationship and whether we can estimate the relative prediction error based of just the S matrix would be useful.

7-2-4 Predicting on a 95% interval

For the most part we kept to estimating $\mathbb{E}\left[\overline{\Gamma(N)}\right]$ which is equivalent to a 50^{th} percentile certainty estimate of total processing time. But in some applications one might want more certainty than that, if for example one might want 2 sigma certainty that would require a 97.7^{th} percentile certainty estimate. This can of course already be calculated using the fitted extreme value distributions, but these become less reliable at the edges of their distributions.

Further research might be done to fit these distributions in such a way that they are more reliable at their extremes and which systems can be fitted such that they are sufficiently reliable. Secondly, it might be possible to adapt the marginal cost model such that it is not fitted to estimate mean processing time but instead an n^{th} percentile certainty estimate.

Bibliography

[1] B. Heidergott, G. J. Olsder, and J. van der Woude, *Max Plus at work: modeling and analysis of synchronized systems: a course on Max-Plus algebra and its applications*. Princeton University Press, 2014, vol. 48.

[2] B. De Schutter and T. J. van den Boom, "Max-plus algebra and max-plus linear discrete event systems: An introduction," in *2008 9th International Workshop on Discrete Event Systems*, IEEE, 2008, pp. 36–42.

[3] M. Akian, R. Bapat, and S. Gaubert, "Max-plus algebra," *Handbook of linear algebra*, vol. 39, 2006.

[4] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, "Synchronization and linearity: An algebra for discrete event systems," 1992.

[5] R. A. Cuninghame-Green, *Minimax algebra*. Springer Science & Business Media, 2012, vol. 166.

[6] J. G. Braker, "Algorithms and applications in timed discrete event systems.," 1995.

[7] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat, "Numerical computation of spectral elements in max-plus algebra," *IFAC Proceedings Volumes*, vol. 31, no. 18, pp. 667–674, 1998.

[8] J. G. Braker and G. J. Olsder, "The power algorithm in max algebra," *Linear Algebra and its Applications*, vol. 182, pp. 67–89, 1993.

[9] M. Umer, U. Hayat, F. Abbas, A. Agarwal, and P. Kitanov, "An efficient algorithm for eigenvalue problem of latin squares in a bipartite min-max-plus system," *Symmetry*, vol. 12, no. 2, p. 311, 2020.

[10] C. Roser, M. Nakano, and M. Tanaka, "A practical bottleneck detection method," in *Proceeding of the 2001 Winter Simulation Conference (Cat. No. 01CH37304)*, IEEE, vol. 2, 2001, pp. 949–953.

[11] T. J. van den Boom and B. De Schutter, "Modelling and control of discrete event systems using switching max-plus-linear systems," *Control engineering practice*, vol. 14, no. 10, pp. 1199–1211, 2006.

[12] ——, "Modeling and control of switching max-plus-linear systems with random and deterministic switching," *Discrete Event Dynamic Systems*, vol. 22, no. 3, pp. 293–332, 2012.

List of Acronyms

MPL	Max-Plus-Linear
SMPL	Switching Max-Plus-Linear
DES	Discrete Event Systems
JIT	Just In Time
GEV	Generalised Extreme Value
PDF	Probability Density Function
CDF	Cumulative Density Function
SQP	Sequential Quadratic Programming