Max-Plus Algebra applied to Supply Chain Scheduling

E.J. Leijenhorst

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Max-Plus Algebra applied to Supply Chain Scheduling

(Dutch title: Max-Plus algebra toegepast op planning van bevoorradingsketens)

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

JUDITH LEIJENHORST

Delft, the Netherlands
April 2017

BSc Thesis APPLIED MATHEMATICS

"Max-Plus Algebra applied to Supply Chain Scheduling"

(Dutch title: "Max-Plus algebra toegepast op planning van bevoorradingsketens)"

JUDITH LEIJENHORST

Delft University of Technology

Supervisor

Dr. J.W. van der Woude

Thesis committee

Dr. N.V. Budko Dr. J.G. Spandaw

April 2017 Delft

CONTENTS

SUMMARY

In this thesis Max-Plus Algebra is discussed. This is an algebraic structure which is useful for modelling scheduling problems. Instead of normal addition and multiplication respectively the operations maximum and addition are used. Together with max-plus algebra another modelling method is described: Heaps of Pieces. This looks like Tetris and is used for the same purpose. Knowing about these methods the main example is introduced: supply chain scheduling for oiltankers. The two methods described earlier are used to find a solution for this problem. Lastly, using simulated data results are produced and explained.

SAMENVATTING

In dit verslag wordt Max-Plus Algebra besproken. Dit is een algebraïsche structuur die nuttig is voor problemen rondom planning. In plaats van normale optelling en vermenigvuldiging, worden er respectievelijk de operaties maximum en optelling gebruikt. Naast max-plus algebra wordt er nog een andere methode beschreven, namelijk Heaps of Pieces. Dit ziet er uit als Tetris en wordt gebruikt voor dezelfde doeleinden. Nu deze twee methoden bekend zijn wordt het belangrijkste voorbeeld geïntroduceerd: het plannen bevoorradingsketens voor olietankers. De twee eerder beschreven methoden worden gebruikt om een oplossing te vinden voor dit probleem. Tenslotte wordt er gebruik gemaakt van gesimuleerde data waarmee resultaten worden geproduceerd en uitgelegd.

PREFACE

When I started this project I wanted it to be about Max-Plus Algebra. Do not worry if you do not know what that is, I will explain it in the next chapter. It is not a type of math taught in the Applied Mathematics Bachelor program. It was Jacob van der Woude who told me about it.

In my first year, during one of our sessions with our mentor group, Jacob showed us something he had worked on. It was a model of a simple railway network, using Max-Plus Algebra. I was immediately interested, since I had a particular interest in the mathematics behind train schedules. Actually, whenever people asked me why I wanted to study math I usually answered: ¨I either am going to become a teacher or I am going to solve all the problems with trains.¨ So when Jacob sent some files afterwards about the things he had talked about, I read them and kept them. This year, while thinking about interesting topics for my Bachelor Project, I thought of those files again, so I sent Jacob an email to make a first appointment. After he gave me some articles I started to work on the project.

Max-Plus Algebra is usually used for modeling scheduling problems, but can also be used to solve certain problems. Over the past few years, Max-Plus Algebra has been frequently used by the Delft Center for Systems and Control (DCSC). It is often used for modeling periodical scheduling like timetables. However, it can also be used in supply chain scheduling. Both will be demonstrated in this thesis.

In the first chapter I will explain what Max-Plus Algebra is; in the second chapter I will say something about the method of Heaps of Pieces, another useful method to look at scheduling problems. In chapter three I will explain the main issue of this paper and also show how to solve it using the explained methods. Finally, in the fourth chapter, I will discuss the results.

I hope you enjoy reading and learn something new about the field Max-Plus Algebra.

Judith Leijenhorst
Delft, April 2017

1

MAX-PLUS ALGEBRA

1.1. BASIC PROPERTIES

First a few definitions and notational matters will be dilated upon, then a few examples will follow.

1.1.1. DEFINITIONS AND NOTATION

In max-plus algebra, the max-plus semiring is used. This semiring $\mathscr{R}_{\max}$ is a semiring over $\mathbb{R}\cup\{-\infty\}$, equipped with maximum and addition. These operations are denoted as $\oplus$ and $\otimes$. Therefore, the semiring $\mathscr{R}_{\max}$ is denoted as $(\mathbb{R}_{\max}, \oplus, \otimes, \varepsilon, e)$. For $a,\ b \in \mathbb{R}_{\max}$, these operations are defined as:

$$a \oplus b := \max\{a, b\} \tag{1.1}$$

$$a \otimes b := a + b \tag{1.2}$$

As one can see, the symbols look similar to the conventional addition and multiplication. [1] As expected they follow the same rules as to the order of operations: multiplication precedes addition. To illustrate this:

$$a \oplus b \otimes c := a \oplus (b \otimes c)$$

For both operations associativity and commutativity holds. For multiplication distributivity also holds. Furthermore, just like in every semiring, $\mathbb{R}_{\max}$ has a zero element, $-\infty$, and a unit element, 0. Common notation for this zero element: $\varepsilon = -\infty$. In some literature $e = 0$ is used for the unit element.

Matrix addition and multiplication are also defined, these are also similar to the matrix operations on $\mathscr{R}$. Matrix addition can be done element-wise:

$$[A \oplus B]_{ij} := A_{ij} \oplus B_{ij} \tag{1.3}$$

Matrix multiplication is, as always, a little more complicated. Let A be a $n \times p$ matrix and B be a $p \times m$ matrix, then:

$$[A \otimes B]_{ij} := \bigoplus_{k=1}^{p} \left(A_{ik} \otimes B_{kj}\right) \tag{1.4}$$

The algebraic power of $a \in \mathbb{R}_{\max}$ is logically defined, it is denoted as:

$$a^{\otimes n} := \underbrace{a \otimes a \otimes \ldots \otimes a}_{n \text{ times}} = n \cdot a$$

Note that this is basically the same as 'normal' matrix multiplication. Using the last equality not only natural numbers can be used, but also other real numbers. For matrices this is, needless to say, not the case. Calculating a power of a matrix can of course only be done with square matrices. Lastly, one other definition should be given; with A being a matrix in $\mathbb{R}_{\max}$:

$$A^* := \bigoplus_{i=0}^{\infty} A^{\otimes i}$$

This is often used, because a lot of max-plus models contain recurrence relations.

[1] Therefore, these operations will be referred to as addition($\oplus$) and multiplication ($\otimes$)

1

1.1.2. EXAMPLES TO THE DEFINITIONS

To get acquainted with the notations, it seems good to start with some easy examples of definitions 1.1, 1.2 and some properties names after that.

$$
\begin{aligned}
3 \oplus 5 &= \max\{3,5\} = 5 \\
1 \oplus \varepsilon &= \max\{1,-\infty\} = 1 \\
5 \otimes 2 &= 5+2 = 7 \\
4 \otimes \varepsilon &= 1 + -\infty = -\infty
\end{aligned}
$$

Matrix computations are a bit more complicated. Let $A = \begin{pmatrix} 0 & \varepsilon \\ 3 & 2 \end{pmatrix}$ and $B = \begin{pmatrix} 4 & 5 \\ 1 & \varepsilon \end{pmatrix}$. Then, recalling definition 1.3 and 1.4:

$$
\begin{aligned}
A \oplus B &= \begin{pmatrix} 0 \oplus 4 & \varepsilon \oplus 5 \\ 3 \oplus 1 & 2 \oplus \varepsilon \end{pmatrix} \\
&= \begin{pmatrix} 4 & 5 \\ 3 & 2 \end{pmatrix} \\
A \otimes B &= \begin{pmatrix} (0 \otimes 4) \oplus (\varepsilon \otimes 1) & (0 \otimes 5) \oplus (\varepsilon \otimes \varepsilon) \\ (3 \otimes 4) \oplus (2 \otimes 1) & (3 \otimes 5) \oplus (2 \otimes \varepsilon) \end{pmatrix} \\
&= \begin{pmatrix} 4 \oplus \varepsilon & 5 \oplus \varepsilon \\ 7 \oplus 3 & 8 \oplus \varepsilon \end{pmatrix} \\
&= \begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix}
\end{aligned}
$$

1.2. EXAMPLE: SIMPLE RAILWAY NETWORK

To illustrate the use of max-plus algebra, the following example is often used. This is an example of synchronized max-plus algebra. Consider a simple train network, with two main stations, A and B, as seen in the picture below, figure 1.1. The labels at the edges represent travel times. There are four trains, one on both local circuits, and two on the inner circuit. Trains should depart as frequent as possible, but regularly. Another condition is that trains should wait for the other train to arrive before it can depart, so passengers have the opportunity to change trains.

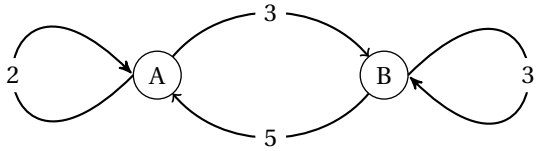

Figure 1.1: A simple railway network

Let x_A represent the departure time of the trains at station A and let x_B do the same for station B. Together this forms a vector $\boldsymbol{x} \in \mathbb{R}^2$. The first departure is $x(0)$, the next one $x(1)$ and so forth. In general, the kth departure time is given by $x(k-1)$. Because of the given criteria, trains from station A can depart if the train on the local track and the train from the inner circuit both have arrived. Therefore the maximum of both times should be taken. The arrival time of the train on a circuit is equal to the time of its last departure plus

the travelling time. With this information the following model can be obtained.

$$\begin{aligned} x_A(k+1) &= \max\{x_A(k)+2,\ x_B(k)+5\} \\ &= (x_A(k)\otimes 2)\oplus(x_B(k)\otimes 5) \\ x_B(k+1) &= \max\{x_A(k)+3,\ x_B(k)+3\} \\ &= (x_A(k)\otimes 3)\oplus(x_B(k)\otimes 3) \end{aligned}$$

This can be written in matrix notation. As one can see, this is a recurrence relation.

$$\boldsymbol{x}(k+1) = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix} \otimes \boldsymbol{x}(k)$$

The obvious solution to this equation is:

$$\boldsymbol{x}(k) = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}^{\otimes k} \otimes \boldsymbol{x}(0)$$

To make this explicit, one could try different vectors for $\boldsymbol{x}(0)$ to see what a suitable starting time would be to make a regular schedule. Another way to do this it is trying to find a $\lambda \in \mathbb{R}$ so that $A \otimes \boldsymbol{x} = \lambda \otimes \boldsymbol{x}$. If such a λ exists, the problem can be solved since $A^{\otimes k} \otimes \boldsymbol{x} = \lambda^k \otimes \boldsymbol{x}$. This will not be explained in this thesis, more about this can for example be found in Max-Plus At Work [1].

2

PETRI NETS AND HEAPS OF PIECES

2.1. PETRI NETS

A Petri net is a graphical way of describing a certain class of discrete event dynamic systems, stepwise processes. It is a directed bipartite graph with two types of nodes: transitions, representing events that may occur, and places. These nodes are connected by arcs abiding by the rule that arcs only run between a place and a transition, never between two places or two transitions. A transition is represented by a bar and places by circles.

Places may contain tokens, available resources. These tokens are shown as dots in a place like in figure 2.1. If there is a token available in all places preceding a transition, the transition fires automatically. Firing means that the input tokens are consumed by the transition and output tokens are created in all its successors. The formal definitions can for example be found in *Time and Petri nets*[2].

There are two types of Petri nets: timed and untimed Petri nets. For a Petri net being timed means that there are *firing times* associated with transitions and/or *holding times* associated with places. In this thesis timed Petri nets will be used, but only with holding times. These holding times are stated after the name of a place. This can for instance be seen in the figure below: $P_2,\ t$ means that P_2 has a holding time of t time units.

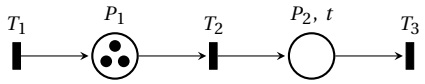

Figure 2.1: A small Petri net consisting of three transitions and two places

2.2. HEAPS OF PIECES

2.2.1. BASIC PROPERTIES

The *Heaps of pieces* model was introduced by Xavier Viennot [3]. In this model solid blocks (*pieces*) are piled up, like the mechanism in the Tetris game, except they canńt be rotated or moved horizontally. Since they cannot move horizontally, the horizontal places the pieces take up are called *slots*. Together these pieces form a *heap*. The set of pieces is denoted by $\mathscr{T}$, the set of slots by $\mathscr{R}$. Stacking these pieces can be used in optimizing processes.

For a piece a, the slots it occupies are indicated by $R(a)$. The lower and upper contour are denoted by $l(a,r)$ and $u(a,r)$ where r is a slot. If $u(a,r) = l(a,r) = -\infty$, then the piece does not occupy that particular slot. The height of a block can be zero, as can be seen in the next example. How this is shown in a graphical representation of the heap can also be seen in the following example.

2.2.2. EXAMPLE

Take, for example, the heap of two blocks in figure 2.2. This model can be described by using the definition given above.

Since there are two pieces and four slots:

$$\mathscr{T} = \{a,\ b\},\ \mathscr{R} = \{1,\ 2,\ 3,\ 4\}$$

In this example piece a occupies all four slots, piece b does not:

$$R(a) = \{1,\ 2,\ 3,\ 4\},\ R(b) = \{1,\ 2,\ 3\}$$

The lower contour of piece a is a straight line, so it has height zero everywhere. The right part of the upper contour can be easily read, the leftmost slot however has height zero. The contour of block b is quite obvious. Writing this in mathematical terms gives:

$$u(a,\cdot) = [0,\ 2,\ 1,\ 1];\ l(a,\cdot) = [0,\ 0,\ 0,\ 0]$$
$$u(b,\cdot) = [2,\ 2,\ 2,\ -\infty];\ l(b,\cdot) = [1,\ 1,\ 0,\ -\infty]$$

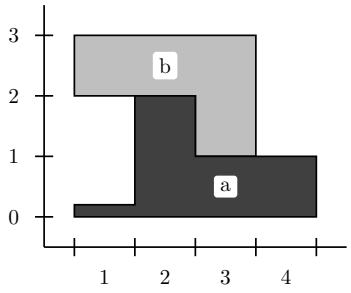

Figure 2.2: Example Heap

2.3. HEAPS OF PIECES OBTAINED FROM PETRI NETS

One nice thing about this kind of model is that a heap can directly be obtained from a Petri net. [4] The pieces in the heap represent the transitions from the Petri net, the resources represent the places. The heights of the pieces can be determined in the following way:

- when a transition a and a place r **are not** connected piece a will not occupy that slot r: $l(a,r) = u(a,r) = -\infty$;
- when a transition a and a place r **are** connected the lower contour of piece a is zero: $l(a,r) = 0$;
- when there is only an outgoing arrow from place r to transition a the upper contour of piece a in slot r is zero: $u(a,r) = 0$;
- when there is an arrow from transition a to place r, the height of piece a equals the holding time $\tau(r)$ of that place: $u(a,r) = \tau(r)$

This can be done for every transition. The heap below can for instance be obtained from the following Petri net:

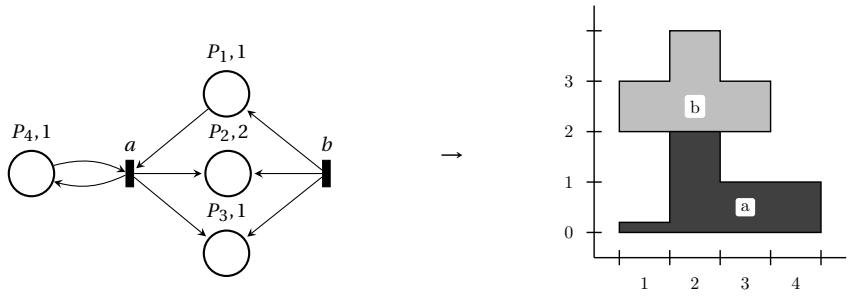

Figure 2.3: Petri net and the corresponding heap

To see how this is done, it is needed to look at the four steps described above, but first note that there are four resources because there are four places. There will be two pieces, because there are two transitions. Now firstly, piece a will occupy all four slots because transition a is connected to all places. Piece b is not connected to P_4, so it will not occupy the fourth slot. Secondly, for the other slots the lower contours of the blocks will be zero; the bottom is flat. Next, the upper contours should be determined. The height of a piece in each slot is equal to the holding time of the corresponding place. This is easily done, but note that there is only an arrow from P_1 to a and not the other way around. Hence, the height of block a in slot 1 should be zero.

3

SUPPLY CHAIN SCHEDULING: TANKERS

The main example of this thesis will be a problem of supply chain scheduling for the distribution of oil. In this problem there are a few things which should be considered. Firstly, there is a number of requests, which include a request date t and the amount of oil needed N_d. Furthermore, there is a number of tankers n which each has a certain capacity C_t. The time needed for transport from the supplier to the consumer is denoted by t_t, the time needed to get back to the distribution centre by t_r. It should be pointed out that a delivery can be early, but it cannot be late. Of course the number of days early should be minimized. The goal is to schedule this optimally, minimizing the costs and thus the days early. This can be done by making the oil available at the right time.

To make the situation more clear, figure 3.1 provides a picture of a Petri net of the system. This Petri net represents a transport system with two tankers, but it can be extended to n tankers. The oil enters the Petri net at transition $u(k)$, after which it is ready to be shipped (P_1). It will leave (x_1) if there is an empty tanker available (P_2). In that case, a token will move to P_3 where it will stay for a certain travel time t_t. After the tanker arrives (x_2), the tankers take some time (t_t) to return to the supplier location and the oil is ready for the consumer (P_4) to be used there ($y(k)$). The functions $u(k)$, $x_1(k)$, $x_2(k)$ and $y(k)$ represent the times the corresponding transition fires.

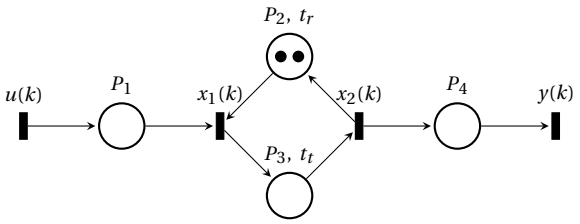

Figure 3.1: Petri net model for the oil tanker example with two tankers

3.1. SOLVING THE PROBLEM USING MAX-PLUS ALGEBRA FOR GENERAL n

Now, a system of equations can be obtained. A filled tanker can leave from the supplier when there is an empty tanker available and there is oil available. For a tanker to be back at the supplier it has to have finished its last trip. All the other tankers will have left too by then, so it is the n^{th} trip after the previous one of that tanker. After it travels the time t_t, it arrives at the consumer. The variable k means that the event happens for the k^{th} time. In terms of max-plus algebra, this can be written as:

$$\begin{aligned} x_1(k) &= x_2(k-n) \otimes t_r \oplus u(k) \\ x_2(k) &= x_1(k) \otimes t_t \\ y(k) &= x_2(k) \end{aligned} \tag{3.1}$$

To simplify this, these equations can first be put into the form of a matrix equation:

$$\boldsymbol{x}(k) = A_0 \otimes \boldsymbol{x}(k) \oplus A_1 \otimes \boldsymbol{x}(k-n) \oplus B_0 \otimes u(k) \tag{3.2}$$
$$y(k) = C \otimes \boldsymbol{x}(k) \tag{3.3}$$

where

$$\boldsymbol{x}(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix},\ A_0 = \begin{bmatrix} \varepsilon & \varepsilon \\ t_t & \varepsilon \end{bmatrix},\ A_1 = \begin{bmatrix} \varepsilon & t_r \\ \varepsilon & \varepsilon \end{bmatrix},\ B_0 = \begin{bmatrix} 0 \\ \varepsilon \end{bmatrix},\ \text{and } C = \begin{bmatrix} \varepsilon & 0 \end{bmatrix}$$

However, equation 3.2 is still implicit. To make it explicit, $\boldsymbol{x}(k)$ is substituted in the right-hand side N times. Since A_0 is nilpotent[1], the first term will vanish. Just like the normal multiplication symbol the $\otimes$-symbol is sometimes omitted to shorten formulas.

$$\begin{aligned}
\boldsymbol{x}(k) &= A_0^{\otimes 2} \otimes \boldsymbol{x}(k) \oplus A_0 \otimes \boldsymbol{x}(k-n) \oplus A_0 A_1 \otimes \boldsymbol{x}(k-n) \oplus B_0 \otimes u(k) \oplus A_0 B_0 \otimes u(k) \\
&= A_0^{\otimes N} \otimes \boldsymbol{x}(k) \oplus \bigoplus_{i=0}^{N-1} \left(A_0^{\otimes i} A_1 \otimes \boldsymbol{x}(k-n) \oplus A_0^{\otimes i} B_0 \otimes u(k) \right) \\
&= \mathcal{E} \otimes \boldsymbol{x}(k) \oplus A_0^* A_1 \otimes \boldsymbol{x}(k-n) \oplus A_0^* B_0 \otimes u(k) \qquad \text{as } N \to \infty \\
&= A \otimes \boldsymbol{x}(k-n) \oplus B \otimes u(k)
\end{aligned} \tag{3.4}$$

where $A = A_0^* A_1$ and $B = A_0^* B_1$; $\mathcal{E}$ stands for the zero-matrix, a matrix filled with epsilons.

Remember that the goal is to find out at what time the orders should be shipped, in order to deliver the oil at the right time. Therefore, the transfer matrix is required, a matrix H so that $Y = H \otimes U$. Here Y and U are vectors consisting of the separate $y(k)$ and $u(k)$. This means that the equation for y can only depend on u, not on $\boldsymbol{x}$. Therefore, $\boldsymbol{x}(k)$ need to be eliminated. Here, $\alpha = \left\lfloor \frac{k}{n} \right\rfloor$.

$$\begin{aligned}
y(k) &= C \otimes \boldsymbol{x}(k) \\
&= CA \otimes \boldsymbol{x}(k-n) \oplus CB \otimes u(k) \\
&= CA^{\otimes \alpha} \otimes x(k - \alpha \cdot n) \oplus \bigoplus_{i=0}^{\alpha-1} \left(C \otimes A^{\otimes i} B \otimes u(k - i \cdot n) \right)
\end{aligned} \tag{3.5}$$

$$\tag{3.6}$$

Since $y(k)$ cannot depend on $\boldsymbol{x}$ now, $\boldsymbol{x}(k - \alpha \cdot n)$ also needs to be substituted. Because $\boldsymbol{x}$ is not defined for negative k, it can be said that it equals the zero-element, $\mathcal{E}$:

$$\begin{aligned}
\boldsymbol{x}(k - \alpha \cdot n) &= A \otimes \boldsymbol{x}(k - \alpha \cdot n - n) \oplus B \otimes u(k - \alpha \cdot n) \\
&= A \otimes \mathcal{E} \oplus B \otimes u(k - \alpha \cdot n)
\end{aligned}$$

Therefore, equation 3.5 can be rewritten as:

$$y(k) = \bigoplus_{i=0}^{\alpha} C \otimes B \otimes A^{\otimes i} \otimes u(k - i \cdot n) \tag{3.7}$$

This results in the matrices Y, H and U that satisfy the equation $Y = H \otimes U$, for every tanker. If the number of departures is n_d and the tankers are denoted by $l = 1 \ldots n$. Furthermore, γ_l is the number of departures for each tanker l. Mathematically said: γ_l is an integer such that $l + \gamma_l n \leq n_d$.

$$Y(l) = \begin{pmatrix} y(l) \\ y(l+n) \\ \vdots \\ y(l+\gamma n) \end{pmatrix},\ H = \begin{pmatrix} CB & \mathcal{E} & \cdots & \mathcal{E} \\ CAB & CB & \cdots & \mathcal{E} \\ \vdots & \vdots & \ddots & \vdots \\ CA_l^{\gamma} B & CA^{\gamma_l - 1} B & \cdots & CB \end{pmatrix},\ U(l) = \begin{pmatrix} y(l) \\ y(l+n) \\ \vdots \\ y(l+\gamma_l n) \end{pmatrix} \tag{3.8}$$

When not all tankers are of equal capacity this solution is not always optimal, since not always the best tanker is chosen for a shipment, just the first to arrive. An example of this can be seen in chapter 4.

[1] Here nilpotent means a certain power of the matrix will become the zero matrix: a matrix filled with the zero element, $\varepsilon = -\infty$

3.2. SOLVING THE PROBLEM USING HEAPS OF PIECES

Another way to solve a problem like this is the method *heaps of pieces*, as explained in chapter 2. From the Petri net in figure 3.1 the following pieces can be obtained, using the method described in section 2.3. As one can see in figure 3.2, the height of the block corresponding to transition x_1 has a height equal to the travel time in slot 3.

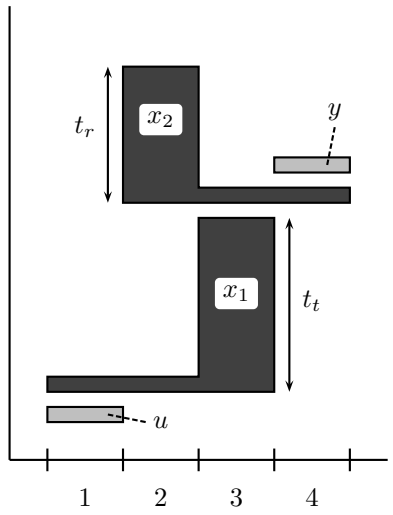

Figure 3.2: Heap model to the Petri net in figure 3.1

Normal use of heaps of pieces would be to stack the pieces from below, like in a Tetris game. However, since in this case the tankers are not allowed to arrive late, stacking should be done backwards. In case of one tanker, the top of the heap would look like this:

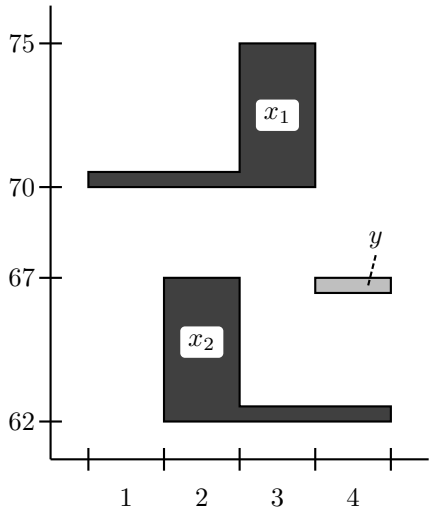

Figure 3.3: Top of the heap of pieces in case of one tanker

The large gap between the two pieces can be explained by the extra restriction that the tankers cannot be late.

4

RESULTS

All the results in this chapter can be obtained using logical reasoning, but it is easier to use max-plus algebra or the heaps of pieces model. In this thesis both methods are used. The heaps of pieces model seems to be optimal in this particular case.

4.1. ONE TANKER, ONE CUSTOMER

Assume that there is one tanker which has to deliver oil to one customer at or before the following dates, taken from [5]: 20-09, 28-09, 06-10, 14-10, 17-10, 23-10, 03-11, 10-11, 18-11, 26-11, 04-12. These are all in the year 2013, which of course does not matter. These dates can be expressed in days from September 20th, as done in table 4.2. Now there are several possibilities to look at. The first is that the capacity of the tanker is big enough for all of the individual requests. The tanker then only has to travel once per request. Let us also look at the possibilities in which case there is one request larger than the capacity of the tanker. In that case the tanker has to travel twice to fulfil the needs of the customer. With the Python code in appendix A this results in:

Demand of request k is larger than capacity tanker	Number of requests early	Number of days early
None	5	15
1	5	22
2	5	29
3	5	36
4	5	43
5	5	50
6	5	57
7	6	40
8	7	47
9	8	46
10	9	44
11	9	41

Table 4.1: Number of days early in case of one tanker where one request is larger than the capacity of the tanker

To see how this number of days early is obtained, one can look at the specifics for the first case: none of the tankers has insufficient capacity. By logical reasoning or using a program table 4.2 can be acquired. Another example is table 4.3. In this case the total number of days early is 40, as can be seen in table 4.1. The trick is to add another request at the same time when the capacity of a tanker is not sufficient, as is the case for the seventh request.

Departure time	Arrival time	Date of demand	Number of days early
-7	-2	0	2
0	5	8	3
7	12	16	4
14	19	24	5
21	26	27	1
28	33	33	0
39	44	44	0
46	51	51	0
54	59	59	0
62	67	67	0
70	75	75	0

Table 4.2: Departure and arrival times in case of one tanker, the tanker capacity fits all demands

Departure time	Arrival time	Date of demand	Number of days early
-10	-5	0	5
-3	2	8	6
4	9	16	7
11	16	24	8
18	23	27	4
25	30	33	3
32	37	44	7
39	44	44	0
46	51	51	0
54	59	59	0
62	67	67	0
70	75	75	0

Table 4.3: Departure and arrival times in case of one tanker; the seventh request is too large for the tanker

4.2. TWO TANKERS, ONE CUSTOMER

In reality, there usually is more than one tanker. In this section the presence of two tankers is discussed, but the method is the same with n tankers. Let us take two tankers, one with a capacity large enough for all requests, one with a capacity smaller than all requests except for one. Again, the data from [5] is used. With the Python code in appendix A this results in:

Demand of request k is less than capacity tanker 1	Number of requests early	Number of days early
None	5	15
1	4	13
2	3	10
3	2	6
4	1	1
5	0	0
6	4	10
7	5	15
8	5	15
9	5	15
10	5	15
11	5	15

5

EXTENSION

The model as described in chapter 3 only takes in account one customer, or multiple customers at the same spot. This, however, is not fully realistic, since there are usually different customers on different spots. In this chapter a situation with two customers will be described, which can be extended to m customers. In figure 5.1 this situation is sketched. Note that place P_1 in this figure can be compared to place P_2 in figure 3.1.

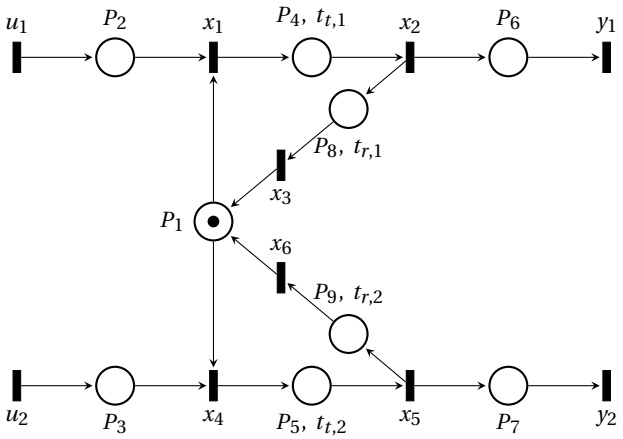

Figure 5.1: Petri net model for the oil tanker example, with two customers and one tankers

In this figure it looks like there is more than one supplier. That is not necessarily the case, it just has to do with the definition of a Petri net.
Using the petri-net, one can see that a ship can leave the suppliers location if there is oil available (transition u_1 or u_2 has been fired) and a ship available (a token at place $P1$). In order for a ship to be available, it has to have delivered it last shipment, and traveled back. Only on ship however needs to be back, so here a minimum operation is needed. Hence, this cannot be solved within Max-Plus algebra. It can be by using heaps of pieces, but this is quite complicated since there are a lot of transitions and places. The pieces to this Petri net are, excluding all the pieces which only have height zero and the slots with only height zero pieces:

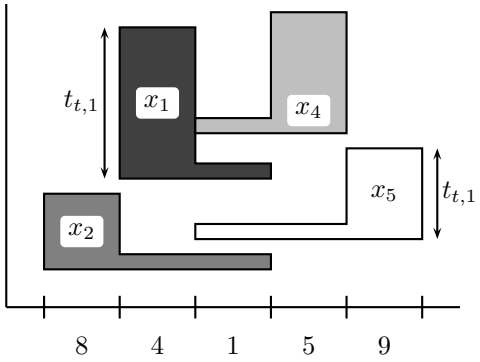

Figure 5.2: Heaps of pieces corresponding to the Petri net in figure 5.1

Another way to solve this would be using Switching Max-Plus Algebra. This is a type of min-max-plus algebra designed to make multiple choices like when to fire u_1 and u_2.

6

CONCLUSION AND DISCUSSION

In a simple railway network we saw the utility of Max-plus algebra. Max-plus algebra makes use of the max-plus semiring, causing the max-plus operations follow the same rules as the 'normal' addition and multiplication.

After the railway example of regular scheduling, an oil tanker supply chain schedule was introduced. A supply chain scheduling like this one can be solved using only max-plus algebra, but there are also other methods like heaps of pieces. This way, the optimal schedule can be chosen. We saw that employing more tankers is not always better; the shipment times are also very important. After looking at a model with one customer, we tried to extend this model to m customers. This problem however cannot be solved within the max-plus algebra. Solving this problem would be a good next step in further research.

Other extensions to include in this research would be taking delays caused by external influences into account. This may be done by using stochastic variables.

BIBLIOGRAPHY

[1] B. Heidergott, G. J. Olsder, and J. van der Woude, *Max Plus at Work* (Princeton University Press, Princeton, New Jersey, 2006).

[2] L. Popova-Zeugmann, *Time and Petri nets* (Springer, 2013).

[3] G. X. Viennot, *Heaps of pieces, i: Basic definitions and combinatorial lemmas,* Annals of the New York Academy of Sciences **576**, 542 (1989).

[4] S. Gaubert and J. Mairesse, *Modeling and analysis of timed petri nets using heaps of pieces,* IEEE transactions on automatic control **44**, 683 (1999).

[5] Subiono and K. Fahim, *On computing supply chain scheduling using max-plus algebra,* Applied Mathematical Sciences **10**, 477 (2016).

A

PYTHON CODE

A.1. PYTHON CODE FOR IMPLEMENTING MAX-PLUS ALGEBRA

```
# Definitions of the maxplus operations
import numpy as np


def plus(a,b):  # summation of a and b; a and b can be numbers or matrices.
if isNumber(a) and isNumber(b):
return max(a,b)
elif isMatrix(a) and isMatrix(b):
if a.shape == b.shape:
n,m = a.shape
c = np.ndarray(shape=a.shape)
for i in range(0,n):
for j in range(0,m):
c[i,j] = max(a[i,j],b[i,j])
return c
else:
print("Please input numbers, either integers or real numbers.")


def times(a,b):  # multiply a and b; a and b can be numbers, matrices or a number and
                                        a matrix.
if isNumber(a) and isNumber(b):
return a + b
elif isMatrix(a) and isMatrix(b):
n,m = a.shape
k,l = b.shape
c = np.ndarray(shape=(n,l))
if m == k:  # matrix dimensions must agree, new array will have size n,l
for i in range(0,n):
for j in range(0,l):
e = eps()
for p in range(0,k):
e = max(e,a[i,p] + b[p,j])
c[i,j] = e
return c
elif isNumber(a) and isMatrix(b):
n,m = a.shape
for i in range(0,n):
for j in range(0,m):
b[i,j] = b[i,j] + a
return b
elif isMatrix(a) and isNumber(b):
return times(b,a)


def power(A,n):  # take the nth power of A, A can either be a matrix or a number
if isMatrix(A) and isNumber(n):
k,l = np.shape(A)
```

A

```
if n > 0 and k == l:
An = A
for i in range(0,n - 1):
An = times(A,An)
return An
elif n == 0:
return epsmat(k)
if isNumber(A) and isNumber(n):
if n > 0:
An = A
for i in range(0,n - 1):
An = times(A,An)
return An
if n == 0:
return 0


def epsmat(n):
A = np.ndarray(shape=(n,n))
A.fill(eps())
np.fill_diagonal(A,0)
return A


def eps():
return -1 * np.inf


def isNumber(a):  # check if a is a number, ie a is a int or a float
return isinstance(a,int) or isinstance(a,float)


def isMatrix(A):  # check if a is a matrix, ie a numpy array
return isinstance(A,np.ndarray)


def solve(A,b):  # Solve Ax=b using min{b_i-a_i,j}
n,m = A.shape
if b.shape == (n,1):
x = np.ndarray(shape=(m,1))
x.fill(np.inf)
for j in range(0,m):
for i in range(0,n):
x[j] = min(x[j],b[i,0] - A[i,j])
return x


def solve2(A,b):  # Solve Ax=b using x = -(-b^T*A)^T
b = -1 * np.transpose(b)
x = times(b,A)
x = -1 * np.transpose(x)
return x
```

A.2. PYTHON CODE FOR IMPLEMENTING AND SOLVING THE OIL TANKER PROBLEM

```
import numpy as np
import maxplus as mp
import xlrd
from datetime import date
import os

e = mp.eps()


def makeH(tt,tr,nd):  #makes matrix H for algorithm 2
C = np.array([[mp.eps(),0]])
B = np.array([[0],[tt]])
A = np.array([[mp.eps(),tr],[mp.eps(),tt + tr]])
H = np.ndarray(shape=(nd,nd))
for i in range(0,nd):
for j in range(0,nd):
if j > i:
H[i,j] = e
else:
An = mp.power(A,(i - j))
CAn = mp.times(C,An)
CAnB = mp.times(CAn,B)
H[i,j] = CAnB
return H


def getValues(book,n):  #reads values from excel
# get the capacity of the tanker
sheet = book.sheet_by_name('Capasity Of Tanker')
Ct = np.empty(n)
for i in range(0,n):
cap = sheet.cell(i,0).value
Ct[i] = cap

# get the number of demand, should be different with more dates
sheet = book.sheet_by_name('Number of BBM per demand')
Nd = np.empty(11,dtype=int)
for i in range(0,11):
Nd[i] = int(sheet.cell(i,1).value)

# get the dates
sheet = book.sheet_by_name('Date of Demand')
dates = np.zeros(11,dtype=date)
for i in range(1,12):
d = int(sheet.cell(i,0).value)
m = int(sheet.cell(i,1).value)
y = int(sheet.cell(i,2).value)
dates[i - 1] = date(y,m,d)
Dd = np.zeros(11,dtype=int)
for i in range(0,11):
Dd[i] = int((dates[i] - dates[0]).days)

return np.sort(Dd,axis=0),Nd,Ct


def alg1(n,Dd,Nd,Ct):  #algorithm 1
y = []
nd = 0
for i in range(0,Dd.size):
temp = Nd[i]
while temp > 0:  #checks if demand > capacity tanker
temp -= Ct[nd % n]
y.append(Dd[i])
nd += 1
return nd,np.array(y)
```

A

```
def alg2(n,tt,tr,Dd,Nd,Ct):
nd,y = alg1(n,Dd,Nd,Ct)
u = np.ndarray(shape=(n,int(nd / n)))
for l in range(0,n):
g = int((nd - l) / n)

H = makeH(tt,tr,g)  #should be different ico more than one tanker
ind = np.arange(g)
Y = (np.take(y,ind * n + l)).reshape((g,1))
x = mp.solve2(H,Y)
u = [(x[i][0],Y[i]) for i in range(g)]
return np.array(u)


def my_early(u,tt):  # note: this calculation is different from subiono's
early = 0
n = np.shape(u)[0]
m = [np.shape(u[x])[0] for x in range(n)]
for i in range(n):
j = m[i] - 1
while j >= 0:
if (j != 0) and (u[i][j - 1][1] == u[i][j][1]):
j -= 1
else:
early += u[i][j][1] - u[i][j][0] - tt
j -= 1
return early


def early(u,tt): #subiono's version of early calculation
early = 0
n = np.shape(u)[0]
m = [0 for x in range(n)]
for i in range(n):
m[i] = np.shape(u[i])[0]
for j in range(m[i]):
early += u[i][j][1] - u[i][j][0] - tt
return early

def heaps2(Dd,Nd,Ct,tt,tr):
n = len(Ct)
Ct = np.sort(Ct,axis=None)

u = [None for x in range(n)]
for i in range(0,n):
u[i] = []

k = len(Dd) - 2
Ci = 0

while Ci < n - 1 and Nd[k] > Ct[Ci]: #finds smallest tanker that fits the demand
Ci += 1
for i in range(n):
u[i] = [(Dd[k + 1] - tt,Dd[k+1])]
if Nd[-1] > Ct[Ci]:
i = 1
temp = Nd[-1] - Ct[Ci]
while temp > 0:
i += 1
temp -= Ct[Ci]
u[Ci].insert(0,(Dd[-1] - i * tt - (i - 1) * tr,Dd[k+1]))  #leaving time

while k >= 0:
Ci = 0
while Ci < n - 1 and Nd[k] > Ct[Ci]:
Ci += 1
temp = Nd[k]
i = 0
while temp > 0:
i += 1
temp -= Ct[Ci]
```

```
if u[Ci][0][0] - tr >= Dd[k]:
u[Ci].insert(0,(Dd[k] - i * tt - (i - 1) * tr,Dd[k]))  #leaving time
else:
u[Ci].insert(0,(u[Ci][0][0] - tr - tt,Dd[k]))
k -= 1

k = len(Dd) - 2
Ci = 0
while Ci < n - 1 and Nd[k] > Ct[Ci]:
Ci += 1
for i in range(n):
if i != Ci:
del(u[i][-1])

for i in range(n):
u[i] = np.array(u[i])

return np.array(u)


def main():
while True:
num = input("How many tankers do you want to use? Please input 1 or 2. ")
for i in range(0,12):
if num == '1' or num == 'one':
n = 1
fileDir = os.path.dirname(os.path.realpath('__file__'))
excelname = 'Demand/Demand 1/Demand one tanker ' + str(i) + '.xls'
filename = os.path.join(fileDir,excelname)
book = xlrd.open_workbook(filename)
elif num == '2' or num == 'two':
n = 2
fileDir = os.path.dirname(os.path.realpath('__file__'))
excelname = 'Demand/Demand 2/Demand two tanker ' + str(i) + '.xls'
filename = os.path.join(fileDir,excelname)
book = xlrd.open_workbook(filename)
else:
print("please input 1 or 2.")
break
tt,tr=5,2
Dd,Nd,Ct = getValues(book,n)
u = alg2(n,tt,tr,Dd,Nd,Ct)
u = heaps2(Dd,Nd,Ct,tt,tr)

for l in range(n):
if u[l] != []:
print("u[" + str(l) + "] = " + str(np.transpose(u[l])[0]))
print("y[" + str(l) + "]= " + str(np.transpose(u[l])[1]))
print("u[" + str(l) + "]= " + str(np.transpose(u[l])[1]-np.transpose(u[l])[0]-tt))
print("early 1 = " + str(my_early(u,tt)))
print("early 2 = " + str(early(u,tt)))
print()
else:
break


main()
```