Spring 5-2014

The Tropical Eigenvalue-Vector Problem from Algebraic, Graphical, and Computational Perspectives

Alex William Nowak
Bates College, anowak@bates.edu

The Tropical Eigenvalue-Vector Problem from Algebraic, Graphical, and Computational Perspectives

Alex Nowak

Department of Mathematics, Bates College, Lewiston, ME 04240

The Tropical Eigenvalue-Vector Problem from Algebraic, Graphical, and Computational Perspectives

An Honors Thesis

Presented to

The Faculty of the Department of Mathematics

Bates College

in partial fulfillment of the requirements for the

Degree of Bachelor of Arts

by

Alex Nowak

Lewiston, Maine

March 21, 2014

Contents

Acknowledgments

The extent to which I must thank those who contributed to the completion of this project stretches far beyond these meager pages, but I suppose I have to start somewhere.

For roughly a year now, my advisor, Professor Catherine Buell, has perpetually refreshed my interest in tropical mathematics; has engaged me with exciting lines of mathematical inquiry; has sat patiently, waiting for me to stop staring blankly at the board in her office and learn something, and has exhibited the utmost attention to my intellectual development. For making the process of rolling up my sleeves and covering a board with $\oplus$'s and $\otimes$'s the best academic experience of my life, I thank you Catherine.

Other members of the Department to whom I owe many thanks include Professor Pallavi Jayawant, who made a wonderful contribution to my exposition of the proof of Karp's Algorithm. Professor Adriana Salerno, a master of Sage, contributed greatly to the programming aspect of this project. Professor Chip Ross had great insights into the writing process in our semi-weekly thesis meetings, and Professor Meredith Greer (along with Pallavi) helped vastly with my transition into the second half of this project by providing invaluable comments on my first-semester talk.

My fellow thesis-writers, Sean and Sara, are also owed much gratitude. Not only were your comments and suggestions during our meetings helpful, but the encouragement displayed as I worriedly paced about the first floor of Ladd and the grounds of the Math and Stats Workshop did not go unnoticed. Ben, I am going to miss our late-night commiserating over the honors process.

Last, but not least, I would like to thank those outside of the Bates community that made this project possible. My parents who, in addition to mortgaging an obscene sum of capital on the cultivation of my intellect, have managed to, not just throughout the process of writing this thesis but my whole life, love me unconditionally. Without that love, I don't know wherefrom the drive and focus to produce this work would have come.

Introduction

This thesis presents a language, a language that articulates an order embedded in systems that appear to behave unmanageably. Suppose satellites in a network relay messages to one another so that people in Kansas can watch sumo wrestling. The act of one satellite communicating to another can be viewed as a discrete event. Satellite 1 prepares its message, transmits it to satellite 2, satellite 2 interprets the message, and thus, the event ends. Furthermore, it is important to note that each of these processes works towards a common goal: the broadcasting of sumo wrestling for the viewing pleasure of many grateful Kansans. When our network consists of many satellites sharing messages at different rates (rates contingent upon variables such as communication distance and the particular ability of each satellite) is when the task of managing this system can become daunting.

Luckily, mathematicians, as they are wont to do, have an abstraction to help. We refer to the kind of setup described above as a discrete event dynamic system, and the algebraic structure often employed to describe these systems is the max-plus semifield, where to "add" means to maximize, and to "multiply" means to add. That is, x "plus" y is equivalent to $\max\{x, y\}$, and x "times" y is equivalent to $x + y$. The max-plus algebra comprises one half of a field of study known as tropical mathematics. The other semifield we consider is the min-plus algebra. As one would expect, x "plus" y is equivalent to $\min\{x, y\}$

in the min-plus framework, while multiplication is also translated into standard addition. We will develop more rigorously the notion of a semifield and the algebraic properties of the max-plus and min-plus structures in Chapter 1. Our first chapter also introduces the theory of graphs. Graphs are structures that allow us to visualize the interactions between the entities of discrete event dynamic systems. Coupled with graph theory, tropical linear algebra, a domain where elements from the min-plus and max-plus semifields become entries in matrices and vectors, can further shed light on the description of discrete event networks as they evolve over time. In fact, our central object of study is linear algebraic. The tropical eigenvalue-vector pair is, as we will see in Chapter 2, they key to maintaining order in networks. Recall from introductory linear algebra that if $\lambda \in \mathbb{R}$ is an eigenvalue of the matrix A with associated eigenvector $\mathbf{v}$, then

$$A\mathbf{v} = \lambda\mathbf{v}.$$

In Chapter 3, we will investigate algorithms for computing the eigenvalues and associated eigenvectors of tropical matrices. Chapter 4 shall complicate the picture by showing the eigenvalue-vector pair to not be the only means of describing stable behavior of discrete systems. A pair of vectors $(\boldsymbol{\eta}, \mathbf{v})$ known as the generalized eigenmode will be shown to belong to a larger class of matrices than the eigenvalue-vector pair.

The ultimate analytical goal of this transition from eigenvalues to eigenmodes will be to evaluate the validity of using the techniques described in Chapter 3 to compute the generalized eigenmode. More specifically, we investigate where Karp's Algorithm breaks down in its

ability to describe the limiting behavior of discrete event dynamic systems not possessing eigenvalues, and how it can be modified to do so.

CHAPTER 1

Preliminaries

Our opening chapter does not, in any direct way, involve itself with the tropical eigenvalue-vector problem. However, if we are to investigate the problem from algebraic, graphical, and computational perspectives, groundwork must be laid in these respective fields. We overview tropical algebraic structure in the first section, and extend this structure to vectors and matrices in the second. Section 3 introduces the theory of graphs, and our fourth section illustrates the intersection of graph theory and tropical linear algebra. The final section introduces an object, the recurrence relation, which will be necessary for the application of discrete event systems introduced in Chapter 2.

1. Structures of The Max-Plus and Min-Plus Algebras

Extensive work has been done with tropical algebraic structure in [**6**]. Here, we introduce the basic framework of the max-plus and min-plus algebras, both of which fall under our umbrella of tropical mathematics. The former is defined in $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$ under the binary operations $\oplus$ and $\otimes$, where for $a, b \in \mathbb{R}_{\max}$,

$$a \oplus b := \max\{a, b\}, \text{ and}$$

$$a \otimes b := a + b.$$

Min-plus algebra is defined in $\mathbb{R}_{\min} = \mathbb{R} \cup \{\infty\}$ under the binary operations $\oplus'$ and $\otimes$; if $a, b \in \mathbb{R}_{\min}$,

$$a \oplus' b := \min\{a, b\}, \text{ and}$$
$$a \otimes b := a + b.$$

EXAMPLE 1.1. Working with $2, 3 \in \mathbb{R}$,

$$3 \oplus 2 := \max\{3, 2\} = 3,$$
$$3 \oplus' 2 := \min\{3, 2\} = 2, \text{and}$$
$$3 \otimes 2 := 3 + 2 = 5.$$

EXAMPLE 1.2. Based on our intuitive understanding of infinity, it makes sense that for any $a \in \mathbb{R}$, $a > -\infty$ and $a < \infty$; thus,

$$a \oplus -\infty = -\infty \oplus a = a, \text{ and}$$
$$a \oplus' \infty = \infty \oplus' a = a.$$

Examples 1.1 and 1.2 give us a basic understanding of tropical arithmetic. However, our goal is to understand the structure of $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ at a more abstract level, so we define one of modern algebra's seminal structures, the field.

DEFINITION 1.1. Let R be a nonempty set, $+$ be a binary operation we will call addition, and $\times$ denote a second binary operation, multiplication. The triple $(R, +, \times)$ is a *field* if all of the following conditions hold:

(1) There exists an element $0 \in R$ such that for $a \in R$, $a + 0 = 0 + a = a$; we call 0 the *additive identity*.

(2) For each $a \in R$, there exists an *additive inverse*, $-a \in R$, such that $a + -a = -a + a = 0$.

(3) There exists an element $1 \in R$ we call the *multiplicative identity* (or *unity*); for all $a \in R$, $a \times 1 = 1 \times a = a$.

(4) For each $a \in R$ where $a \neq 0$, there exists a *multiplicative inverse* (or *unit*), a^{-1}, such that $a \times a^{-1} = a^{-1} \times a = 1$.

(5) The operations of addition and multiplication are *associative* in R, so that for $a, b, c \in R$, $(a + b) + c = a + (b + c)$, and $(a \times b) \times c = a \times (b \times c)$.

(6) Addition and multiplication are *commutative* in R, or $a + b = b + a$, and $a \times b = b \times a$ for $a, b \in R$.

(7) Multiplication *distributes* across addition; for each $a, b, c \in R$ $a \times (b + c) = a \times b + a \times c$.

In fact, $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ are not fields. We will forgo the tediousness of proving this claim, but let us see, on an informal level, how many of the above criteria are satisfied in $(\mathbb{R}_{\max}, \oplus, \otimes)$ and $(\mathbb{R}_{\min}, \oplus', \otimes)$.

From Example 1.2, we have that the additive identity of $\mathbb{R}_{\max}$ is $-\infty$, and that of $\mathbb{R}_{\min}$ is ∞. We denote these identities with ϵ; it will be clear from context whether we mean ∞ or $-\infty$. The classical analogue to these additive identities is zero, and just as zero times any number is zero, $\epsilon \otimes a = a \otimes \epsilon = \epsilon$ for all $a \in \mathbb{R}_{\max}$ or $a \in \mathbb{R}_{\min}$.

Now consider $a, b \in \mathbb{R}$. Since the set of real numbers is totally ordered either $a \leq b$, or $a \geq b$. Hence, $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ cannot contain additive inverses, as either $a \oplus b = a$, or $a \oplus b = b$, and either $a \oplus' b = a$, or $a \oplus' b = b$. In other words, $a \oplus b \neq \infty$, and $a \oplus' b \neq -\infty$ for all $a, b \in \mathbb{R}$, so neither $\mathbb{R}_{\max}$ nor $\mathbb{R}_{\min}$ have additive inverses.

However, we do have a multiplicative identity. In fact, the multiplicative identity of both $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ is 0. Clearly, $a \otimes 0 = 0 \otimes a = a$

for any $a \in \mathbb{R}$.

To identify tropical multiplicative inverses, we look to $\mathbb{R}$ under standard addition. For any $a \in \mathbb{R}$, its additive inverse is $-a$. Since we define tropical multiplication as standard addition, the multiplicative inverse of $a \neq \epsilon$ contained in $\mathbb{R}_{\max}$ or $\mathbb{R}_{\min}$ is $-a$.

It is relatively easy to see that the operations $\oplus$ and $\oplus'$ are both associative and commutative, an an expression of the form $a \leq b \leq c$ preserves its order, no matter which terms we consider first, and $a \leq b$ implies $b \geq a$. Moreover, since standard addition is both associative and commutative on $\mathbb{R}$, it follows that $\otimes$ is associative and commutative on $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$.

From the following numerical example in $\mathbb{R}_{\max}$, we see that tropical multiplication distributes across tropical addition:

$$\begin{aligned} 4 \otimes (3 \oplus 7) &= 4 \otimes 7 \\ &= 11, \end{aligned}$$

while

$$\begin{aligned} (4 \otimes 3) \oplus (4 \otimes 7) &= 7 \oplus 11 \\ &= 11. \end{aligned}$$

Obviously, we would find similar results working in $\mathbb{R}_{\min}$.

Under their respective additive and multiplicative operations, $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ satisfy all conditions of a field, except for the existence of additive inverses. Structures of this type are called *semifields*, so the triples $(\mathbb{R}_{\max}, \oplus, \otimes)$ and $(\mathbb{R}_{\min}, \oplus', \otimes)$ are semifields.

We will now touch upon other pertinent algebraic properties of $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$. First, we define the notion of idempotent operations.

DEFINITION 1.2. We say that a binary operation, $*$, is *idempotent* on a set R if $a * a = a$ for all $a \in R$.

The operations $\oplus$ and $\oplus'$ are idempotent in $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$, respectively.

We conclude this section with an example that illustrates that tropical algebra allows for the proverbial "freshman's dream", $(x + y)^2 = x^2 + y^2$.

EXAMPLE 1.3. Consider the the following expressions in $\mathbb{R}_{\max}$:

$$\begin{aligned}(3 \oplus 2)^{\otimes 2} &= 3^{\otimes 2} \\ &= 3 \otimes 3 \\ &= 6,\end{aligned}$$

and

$$\begin{aligned}3^{\otimes 2} \oplus 2^{\otimes 2} &= (3 \otimes 3) \oplus (2 \otimes 2) \\ &= 6 \oplus 4 \\ &= 6.\end{aligned}$$

2. Tropical Linear Algebra

So that we may examine the eigenvector-value problem through a tropical framework, we extend the algebraic properties of $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ to matrices and vectors. That is, let $\mathbb{R}_{\max}^{n\times n}$ be the set of $n \times n$ matrices with coefficients in $\mathbb{R}_{\max}$, and let $\mathbb{R}_{\min}^{n\times n}$ be the set of $n \times n$ matrices with coefficients in $\mathbb{R}_{\min}$. Why we only consider square

matrices will become clear in Section 4. We endow each of these sets with the operations of addition and multiplication. Addition is just like matrix addition in the classical sense, but instead of component-wise addition of the elements of each matrix, we take the component-wise maximum or minimum. For $A, B \in \mathbb{R}_{\max}^{n\times n}$ and $C, D \in \mathbb{R}_{\min}^{n\times n}$

$$[A \oplus B]_{ij} = a_{ij} \oplus b_{ij}, \text{ and } [C \oplus' D]_{ij} = c_{ij} \oplus' d_{ij}.$$

Furthermore, let $\otimes$ denote matrix multiplication. The algorithm for tropical matrix multiplication is nearly identical to the classical analogue; each element of the tropical product matrix is the dot product of the corresponding rows and columns. Symbolically speaking,

$$[A \otimes B]_{ij} = \bigoplus_{k=1}^{n} (a_{ik} \otimes b_{kj}) = (a_{i1} \otimes b_{1j}) \oplus (a_{i2} \otimes b_{2j}) \oplus \cdots \oplus (a_{in} \otimes b_{nj}),$$

for $A, B \in \mathbb{R}_{\max}^{n\times n}$. If $C, D \in \mathbb{R}_{\min}^{n\times n}$, then

$$[C \otimes D]_{ij} = \bigoplus_{k=1}^{n}{}' (c_{ik} \otimes d_{kj}) = (c_{i1} \otimes d_{1j}) \oplus' (c_{i2} \otimes d_{2j}) \oplus' \cdots \oplus' (c_{in} \otimes d_{nj}).$$

We denote the multiplication of a matrix A by itself k times by $A^{\otimes k}$.

Let $A = \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}$ and $B = \begin{bmatrix} 2 & 1 \\ 10 & 7 \end{bmatrix}$. Consider the following examples of tropical matrix arithmetic using these two matrices.

EXAMPLE 1.4. If we let $A, B \in \mathbb{R}_{\max}^{2\times 2}$, then

$$\begin{aligned}\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \oplus \begin{bmatrix} 2 & 1 \\ 10 & 7 \end{bmatrix} &= \begin{bmatrix} 8 \oplus 2 & 6 \oplus 1 \\ 4 \oplus 10 & 3 \oplus 7 \end{bmatrix} \\ &= \begin{bmatrix} 8 & 6 \\ 10 & 7 \end{bmatrix}.\end{aligned}$$

EXAMPLE 1.5. Now, with $A, B \in \mathbb{R}_{\min}^{2\times 2}$,

$$\begin{aligned}\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \oplus' \begin{bmatrix} 2 & 1 \\ 10 & 7 \end{bmatrix} &= \begin{bmatrix} 8 \oplus' 2 & 6 \oplus' 1 \\ 4 \oplus' 10 & 3 \oplus' 7 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}.\end{aligned}$$

EXAMPLE 1.6. Multiplying A and B in $\mathbb{R}_{\max}^{2\times 2}$ we find

$$\begin{aligned}\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 2 & 1 \\ 10 & 7 \end{bmatrix} &= \begin{bmatrix} (8 \otimes 2) \oplus (6 \otimes 10) & (8 \otimes 1) \oplus (6 \otimes 7) \\ (4 \otimes 2) \oplus (3 \otimes 10) & (4 \otimes 1) \oplus (3 \otimes 7) \end{bmatrix} \\ &= \begin{bmatrix} 10 \oplus 16 & 9 \oplus 13 \\ 6 \oplus 13 & 5 \oplus 10 \end{bmatrix} \\ &= \begin{bmatrix} 16 & 13 \\ 13 & 10 \end{bmatrix}.\end{aligned}$$

EXAMPLE 1.7. With $A, B \in \mathbb{R}_{\min}^{2\times 2}$

$$\begin{aligned}\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 2 & 1 \\ 10 & 7 \end{bmatrix} &= \begin{bmatrix} (8\otimes 2)\oplus'(6\otimes 10) & (8\otimes 1)\oplus'(6\otimes 7) \\ (4\otimes 2)\oplus'(3\otimes 10) & (4\otimes 1)\oplus'(3\otimes 7) \end{bmatrix} \\ &= \begin{bmatrix} 10\oplus' 16 & 9\oplus' 13 \\ 6\oplus' 13 & 5\oplus' 10 \end{bmatrix} \\ &= \begin{bmatrix} 10 & 9 \\ 6 & 5 \end{bmatrix}.\end{aligned}$$

Another operation to consider is scalar multiplication, which like standard scalar multiplication, is component-wise. For $r \in \mathbb{R}_{\max}$ and $A \in \mathbb{R}_{\max}^{n\times n}$

$$[r \otimes A]_{ij} = r \otimes a_{ij}.$$

The same holds for scalars in $\mathbb{R}_{\min}$ and matrices in $\mathbb{R}_{\min}^{n\times n}$.

EXAMPLE 1.8. Multiplication of A by the scalar 2 in the max-plus and min-plus structures is represented simply as

$$\begin{aligned} 2 \otimes \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} &= \begin{bmatrix} 2\otimes 8 & 2\otimes 6 \\ 2\otimes 4 & 2\otimes 3 \end{bmatrix} \\ &= \begin{bmatrix} 10 & 8 \\ 6 & 5 \end{bmatrix}.\end{aligned}$$

The sets $\mathbb{R}_{\max}^{n\times n}$ and $\mathbb{R}_{\min}^{n\times n}$ contain additive and multiplicative identity elements. Recall that ϵ denotes our additive identities in $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$.

(1) Additive identity in $\mathbb{R}_{\max}^{n\times n}$ and $\mathbb{R}_{\min}^{n\times n}$: $\mathbf{0} = \begin{bmatrix} \epsilon & \epsilon & \cdots & \epsilon \\ \epsilon & \epsilon & \cdots & \epsilon \\ \vdots & \vdots & \ddots & \vdots \\ \epsilon & \epsilon & \cdots & \epsilon \end{bmatrix}$;

(2) Multiplicative identity in $\mathbb{R}_{\max}^{n\times n}$ and $\mathbb{R}_{\min}^{n\times n}$: $E = \begin{bmatrix} 0 & \epsilon & \cdots & \epsilon \\ \epsilon & 0 & \cdots & \epsilon \\ \vdots & \vdots & \ddots & \vdots \\ \epsilon & \epsilon & \cdots & 0 \end{bmatrix}$.

EXAMPLE 1.9. Adding A (from Examples 1.4 through 1.7) to $\mathbf{0}$, we find

$$\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \oplus \begin{bmatrix} \epsilon & \epsilon \\ \epsilon & \epsilon \end{bmatrix} = \begin{bmatrix} 8 \oplus \epsilon & 6 \oplus \epsilon \\ 4 \oplus \epsilon & 3 \oplus \epsilon \end{bmatrix} = \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}.$$

EXAMPLE 1.10. Multiplying A by E yields

$$\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 0 & \epsilon \\ \epsilon & 0 \end{bmatrix} = \begin{bmatrix} (8 \otimes 0) \oplus (6 \otimes \epsilon) & (8 \otimes \epsilon) \oplus (6 \otimes 0) \\ (4 \otimes 0) \oplus (3 \otimes \epsilon) & (4 \otimes \epsilon) \oplus (3 \otimes 0) \end{bmatrix} = \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}.$$

Perhaps most important for our work in later sections is the multiplication of tropical matrices by vectors and the multiplication of vectors by scalars, as these operations are the arithmetic essence of the eigenvalue-vector problem. Let $A \in \mathbb{R}_{\max}^{n\times n}$, $B \in \mathbb{R}_{\min}^{n\times n}$, $\mathbf{x} \in \mathbb{R}_{\max}^{n\times 1}$, $\mathbf{y} \in \mathbb{R}_{\min}^{n\times 1}$, $r \in \mathbb{R}_{\max}$, and $s \in \mathbb{R}_{\min}$. We define the multiplication of matrices and vectors by

$$[A \otimes \mathbf{x}]_i = \bigoplus_{k=1}^{n} a_{ik} \otimes x_k, \text{ and } [B \otimes \mathbf{y}]_i = \bigoplus_{k=1}^{n}{}' b_{ik} \otimes y_k.$$

Scalar multiplication is represented as

$$[r \otimes \mathbf{x}]_i = r \otimes x_i, \text{ and } [s \otimes \mathbf{x}]_i = s \otimes y_i.$$

For the following examples, set $A = \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 2 \\ 10 \end{bmatrix}$.

EXAMPLE 1.11. With $A \in \mathbb{R}_{\max}^{n\times n}$ and $\mathbf{x} \in \mathbb{R}_{\max}^{n\times 1}$

$$\begin{aligned} \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 10 \end{bmatrix} &= \begin{bmatrix} (8 \otimes 2) \oplus (6 \otimes 10) \\ (4 \otimes 2) \oplus (3 \otimes 10) \end{bmatrix} \\ &= \begin{bmatrix} 10 \oplus 16 \\ 6 \oplus 13 \end{bmatrix} \\ &= \begin{bmatrix} 16 \\ 13 \end{bmatrix}. \end{aligned}$$

EXAMPLE 1.12. Working in the min-plus framework, that is, $A \in \mathbb{R}_{\min}^{n\times n}$ and $\mathbf{x} \in \mathbb{R}_{\min}^{n\times 1}$, we find

$$\begin{aligned} \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 10 \end{bmatrix} &= \begin{bmatrix} (8 \otimes 2) \oplus' (6 \otimes 10) \\ (4 \otimes 2) \oplus' (3 \otimes 10) \end{bmatrix} \\ &= \begin{bmatrix} 10 \oplus' 16 \\ 6 \oplus' 13 \end{bmatrix} \\ &= \begin{bmatrix} 10 \\ 6 \end{bmatrix}. \end{aligned}$$

We conclude this section with an example illustrating the intersection of matrix and scalar multiplication in expressing max-plus eigenvectors and values.

EXAMPLE 1.13. Consider the fact that in $\mathbb{R}_{\max}^{n\times n}$

$$\begin{aligned}\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 16 \\ 12 \end{bmatrix} &= \begin{bmatrix} (8 \otimes 16) \oplus (6 \otimes 12) \\ (4 \otimes 16) \oplus (3 \otimes 12) \end{bmatrix} \\ &= \begin{bmatrix} 24 \oplus 18 \\ 20 \oplus 15 \end{bmatrix} \\ &= \begin{bmatrix} 24 \\ 20 \end{bmatrix} \\ &= 8 \otimes \begin{bmatrix} 16 \\ 12 \end{bmatrix}.\end{aligned}$$

The scalar 8 is an eigenvalue of the matrix $\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}$, while $\begin{bmatrix} 16 \\ 12 \end{bmatrix}$ is an eigenvector of the corresponding eigenspace. We will eventually provide a more thorough and formal explanation of tropical eigenvalues and vectors. To lay the foundation for such work, we introduce the theory of graphs.

3. Graph Theory

Underlying many of the max-plus applications we will later visit are graph theoretic concepts. Thus, we provide a brief overview of graphs and their basic properties. For an exposition notationally consistent with that which appears below, see [**5**].

Intuitively speaking, a graph is a set of points with lines connecting some subset of those points.

DEFINITION 1.3. A *graph* G is an ordered triple $G = (V(G), E(G), I_G)$, where $V(G)$, is a nonempty set consisting of *vertices* (or *nodes*), $E(G)$ is a set, whose elements are called *edges*, disjoint from $V(G)$, and I_G

is a relation that associates with each element of $E(G)$ an unordered pair of elements from $V(G)$. The set $V(G)$ is called the *vertex set* of G, and $E(G)$ is called the *edge set*.

We provide examples of graphs below.

EXAMPLE 1.14. Consider the graph $G = (\{1, 2, 3\}, \emptyset, \emptyset)$(pictured in Figure 1.1). Here, we see that only the vertex set need be nonempty.

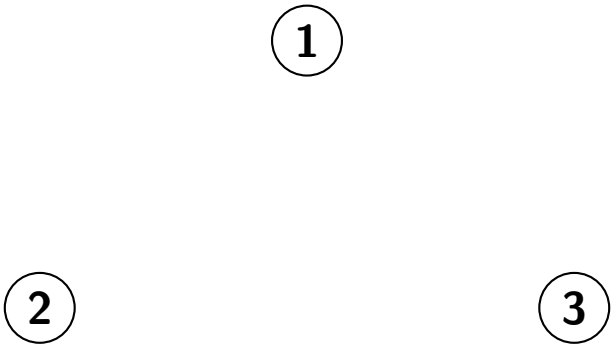


Figure 1.1

EXAMPLE 1.15. We now look at a graph with edges. Represented in Fig 1.2 is $G = (\{1, 2, 3, 4, 5\}, \{e_1, e_2, e_3, \ldots, e_{10}\}, I_G)$, where I_G is given by $I_G(e_1) = \{1, 2\}$, $I_G(e_2) = \{1, 3\}$, $I_G(e_3) = \{1, 4\}$, $I_G(e_4) = \{1, 5\}$, $I_G(e_5) = \{2, 3\}$, $I_G(e_6) = \{2, 4\}$, $I_G(e_7) = \{2, 5\}$, $I_G(e_8) = \{3, 4\}$, $I_G(e_9) = \{3, 5\}$, and $I_G(e_{10}) = \{4, 5\}$.

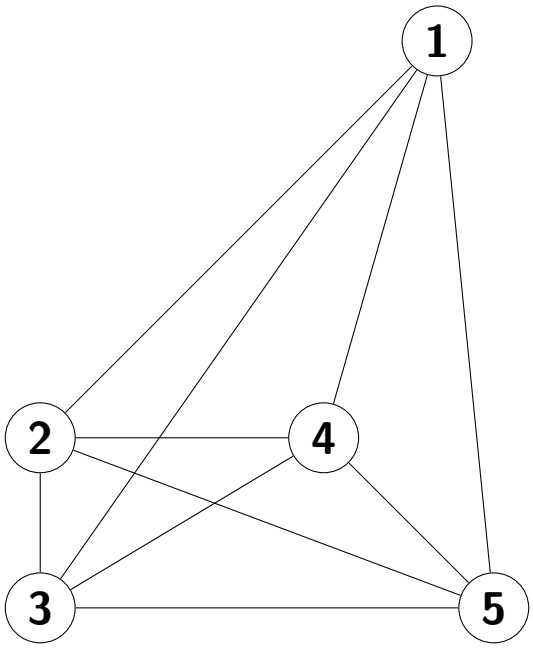


Figure 1.2

As we stated in the introduction, we will consider applications of tropical linear algebra to networks. From the above definition, we see that graphs can be useful for representing networks. Each vertex is an entity in the network, and edges establish connections between these entities. However, we wish to develop a notion of direction in the edges of the graphs we study. There is, in fact, such an object that formalizes direction:

Definition 1.4. A *directed graph*, or *digraph*, is an ordered triple $D = (V(D), A(D), I_D)$, where $V(D)$ is a nonempty *vertex set*, $A(D)$ is a set of *arcs* (also called edges) disjoint from $V(D)$, and I_D is an *incidence map* that associates with each arc in $A(D)$ an ordered pair of vertices from $V(D)$.

We provide an example of a digraph.

EXAMPLE 1.16. The digraph given by $D = (\{1, 2, 3\}, \{a_1, a_2, a_3, a_4, a_5\}, I_D)$, where we define $I_D(a_1) = (1, 1)$, $I_D(a_2) = (1, 3)$, $I_D(a_3) = (2, 1)$, $I_D(a_4) = (3, 1)$, $I_D(a_5) = (3, 2)$ is drawn below in Figure 1.3.

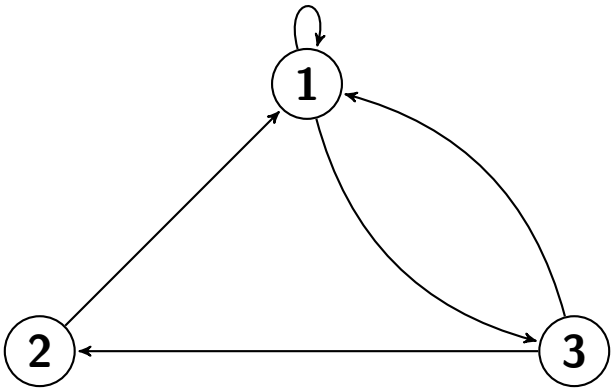


Figure 1.3

Imagine that the above digraph represents a railway network. We have our stations (the nodes), names for the various lines (the arcs), and the direction each line travels (given by our ordered pairs). To get a better sense of the dynamics of our railway system, however, we would want to know the distance between each station. Next, we establish this abstract notion of distance with a special type of digraph.

DEFINITION 1.5. We define a *weighted digraph*, D, as a quadruple $(V(D), A(D), I_D, K_D)$. The set $V(D)$ contains the vertices of D, $A(D)$ (disjoint from $V(D)$) its arcs, and I_D is an incidence map assigning the arcs of $A(D)$ to an ordered pair of vertices in $V(D)$. Finally, K_D is a map that assigns to each arc a real number.

EXAMPLE 1.17. In Figure 1.4, is the weighted directed graph $D' = (\{1, 2, 3, 4, 5\}, \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}, I_{D'}, K_{D'})$. We define $I_{D'}$ by $I_{D'}(a_1) = (1, 2)$, $I_{D'}(a_2) = (1, 3)$, $I_{D'}(a_3) = (2, 3)$, $I_{D'}(a_4) = (2, 4)$, $I_{D'}(a_5) = (2, 5)$, $I_{D'}(a_6) = (3, 5)$, and $I_{D'}(a_7) = (4, 5)$, while $K_{D'}$ is defined by

$K_{D'}(a_1) = 4$, $K_{D'}(a_2) = 5$, $K_{D'}(a_3) = 3$, $K_{D'}(a_4) = 3$, $K_{D'}(a_5) = 4$, $K_{D'}(a_6) = 7$, $K_{D'}(a_7) = 6$.

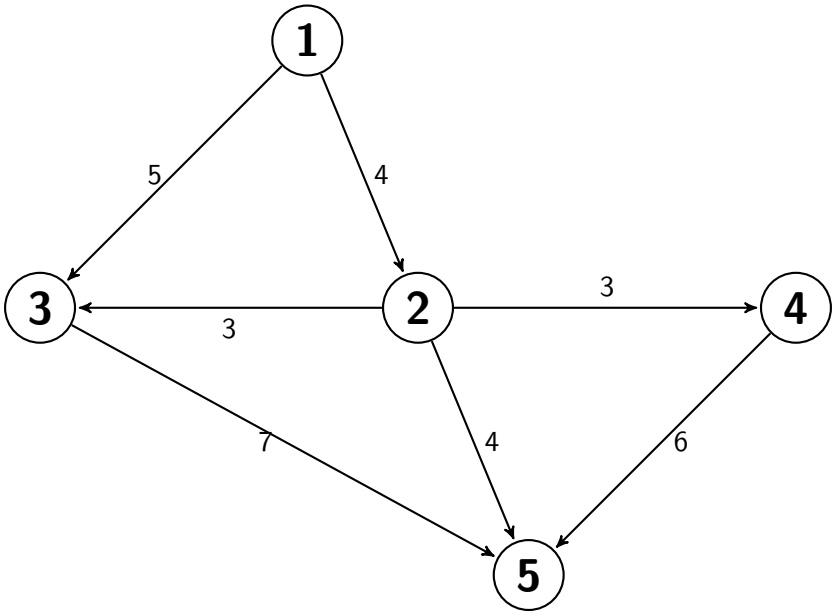


Figure 1.4

Weighted digraphs, such as the one depicted above, will be crucial in our exploration of the tropical eigenvalue-vector problem, but before proceeding, we must explore some of the objects that relate a graph's nodes to its arcs.

Definition 1.6. A *path* in a digraph, D, is a concatenation of distinct edges often denoted by $\rho : e_1 \circ e_2, \ldots, e_p$. Note that the endpoints of consecutive edges must match. In other words, given e_k and e_{k+1} in ρ, if $I_D(e_k) = (v_{q-1}, v_q)$, then $I_D(e_{k+1}) = (v_q, v_{q+1})$. The *path length*, denoted $|\rho|_1$, is equal to the number of edges in the path. The *path weight*, denoted $|\rho|_W$, is equal to the sum of the weights of each edge in the path. We refer to the quantity $\frac{|\rho|_W}{|\rho|_1}$ as the *average* or *normalized* path weight of ρ. If, for $I_D(e_1) = (v_1, v_2)$ and $I_D(e_p) = (v_{q-1}, v_q)$, we

have $v_1 = v_q$, then the path is called a *circuit* (or *cycle*) of D. We typically denote circuits by γ.

REMARK. For our purposes, arcs in digraphs are unidirectional only.

DEFINITION 1.7. Let D be a weighted digraph. If there is a path from any vertex to any other vertex, then D is *strongly connected*.

Consider the graph given in Example 1.16. Some of the cycles of this graph are $\gamma_1 : a_1$; $\gamma_2 : a_2 \circ a_4$; $\gamma_3 : a_2 \circ a_5 \circ a_3$ and $\gamma_4 : a_3 \circ a_2 \circ a_5$. These cycles illustrate the fact that from any node, we can reach any other node in D; thus, D is a strongly connected graph. Our graph from Example 1.17, D', is, however, not strongly connected. For instance, there is no sequence that can begin at node 2 and end at node 1.

4. Matrix Representations of Graphs

First, note that, henceforth, the only graphs we encounter will be weighted digraphs, and it should be assumed that any "graph" we refer to is of the aforementioned type. Furthermore, we are now adopting the convention of referring to all graphs by G; we used D in the previous section to aid the reader in distinguishing between the variety of types presented therein.

Representing graphs with sets and mappings is, as the previous section illustrates, a cumbersome convention. Matrices provide us with a more convenient means of denoting graphs with weighted, directed edges. The translation from graph to tropical matrix is rather simple. We represent the graph G by a matrix A, where the entry a_{ij} denotes the weight of the arc from node j to node i. If two nodes are not

directly linked by an arc, then the corresponding matrix entry is our tropical additive identity, ϵ. We may, thus, also refer to such a graph by $G(A)$. Let us see how such a translation works.

EXAMPLE 1.18. Let $G(A)$ be the graph pictured below.

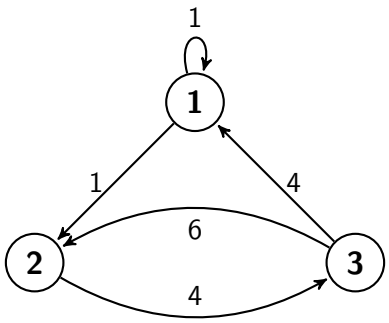


Figure 1.5

Since $G(A)$ has three nodes, A will be a 3×3 matrix. Node 1 sends an arc of weight 1 to itself, so $a_{11} = 1$, and because there is no arc sent from node 2 to node 1, $a_{12} = \epsilon$. On the whole, we have

$$A = \begin{bmatrix} 1 & \epsilon & 4 \\ 1 & \epsilon & 6 \\ \epsilon & 4 & \epsilon \end{bmatrix}.$$

Now that we can translate between graphs and matrices, we can also translate their properties.

DEFINITION 1.8. A tropical matrix A is *irreducible* if and only if its corresponding graph, $G(A)$, is strongly connected. If A is not irreducible, then it is a *reducible* matrix.

Suppose now we let $A \in \mathbb{R}_{\min}^{3\times 3}$. Iterations of the form $A^{\otimes k}$, where $k \in \mathbb{N}$ can actually reveal to us some important features of the graph $G(A)$. More specifically, the matrix entry $[A^{\otimes k}]_{ij}$ gives the path of least weight of length k from node j to node i in $G(A)$. In order to illustrate the power of tropical matrix multiplication, we introduce the "shortest path problem" in the next example.

EXAMPLE 1.19. Suppose Fig. 1.5 is a graphical representation of a railway network between three cities; the arc weights give the times of travel between stations (the path from node 1 to itself is a "sight-seeing" train around city 1). In such an application, our technique of multiplying A by itself to find paths of least weight provides a method for finding the quickest ways to navigate the railway system. To find the fastest means of travel between the three cities, we calculate $A^+ = A \oplus' A^{\otimes 2} \oplus' A^{\otimes 3}$. We compute our multiplicative iterations below:

$$A^{\otimes 2} = \begin{bmatrix} 1 & \epsilon & 4 \\ 1 & \epsilon & 6 \\ \epsilon & 4 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 1 & \epsilon & 4 \\ 1 & \epsilon & 6 \\ \epsilon & 4 & \epsilon \end{bmatrix}$$

$$
\begin{aligned}
&= \begin{bmatrix} 2 \oplus' \epsilon \oplus' \epsilon & \epsilon \oplus' \epsilon \oplus' 8 & 5 \oplus' \epsilon \oplus' \epsilon \\ 2 \oplus' \epsilon \oplus' \epsilon & \epsilon \oplus' \epsilon \oplus' 10 & 5 \oplus' \epsilon \oplus' \epsilon \\ \epsilon \oplus' 5 \oplus' \epsilon & \epsilon \oplus' \epsilon \oplus' \epsilon & \epsilon \oplus' 10 \oplus' \epsilon \end{bmatrix} \\
&= \begin{bmatrix} 2 & 8 & 5 \\ 2 & 10 & 5 \\ 5 & \epsilon & 10 \end{bmatrix}; \\
A^{\otimes 3} &= \begin{bmatrix} 2 & 8 & 5 \\ 2 & 10 & 5 \\ 5 & \epsilon & 10 \end{bmatrix} \otimes \begin{bmatrix} 1 & \epsilon & 4 \\ 1 & \epsilon & 6 \\ \epsilon & 4 & \epsilon \end{bmatrix} \\
&= \begin{bmatrix} 3 \oplus' 9 \oplus' \epsilon & \epsilon \oplus' \epsilon \oplus' 9 & 6 \oplus' 14 \oplus' \epsilon \\ 3 \oplus' 11 \oplus' \epsilon & \epsilon \oplus' \epsilon \oplus' 9 & 6 \oplus' 16 \oplus' \epsilon \\ 6 \oplus' \epsilon \oplus' \epsilon & \epsilon \oplus' \epsilon \oplus' 14 & 9 \oplus' \epsilon \oplus' \epsilon \end{bmatrix} \qquad (1) \\
&= \begin{bmatrix} 3 & 9 & 6 \\ 3 & 9 & 6 \\ 6 & 14 & 9 \end{bmatrix}.
\end{aligned}
$$

We note some illustrative aspects of the above computations. First, we can see from (1) just how min-plus matrix multiplication yields quickest paths. Consider the calculation of $[A^{\otimes 3}]_{13}$, where we sum $6 \oplus' 14 \oplus' \epsilon$. This first term, 6, represents the 3-path route by which we take the train from city 3 to city 1 and then take the city 1 train twice. The term 14 is another 3-path option; here, we would take the train from city 3 to city 2, go back to city 3, and then travel up to city 1. Our third option is infinite ($a_{33} = \epsilon$), as there is no 1-length path from city 3 to itself, and thus no 3-length path with a subpath going from 3 to itself. Min-plus multiplication (understood classically as summing the various arc weights of the graph) gave us these options. Then,

the min-plus addition structure within min-plus matrix multiplication allows us to choose the minimum path from these options. Hence, $a_{13} = 6 \oplus' 14 \oplus' \epsilon = 6$. Second, note $[A^{\otimes 2}]_{32} = \epsilon$. One can easily see by referring to the graph in Fig. 1.5 that there is no 2-length path from node 2 to node 3. Just as infinite values occur in our A matrix when nodes do not directly communicate, $[A^{\otimes 2}]_{ij} = \epsilon$ when node i cannot be reached from j in a path of length 2.

The above explanation of the computation of $[A^{\otimes 3}]_{13}$ illustrates the necessity of finding A^+ to actually solve the shortest path problem. We found the minimum-weight path of length 3 from node 3 to node 1, but why would we want to take the train to city 1 and then proceed to take the same line twice in a row? We want to find the quickest paths through the system, regardless of path length, so we employ min-plus matrix addition to find $A \oplus' A^{\otimes 2} \oplus' A^{\otimes 3}$:

$$\begin{bmatrix} 1 & \epsilon & 4 \\ 1 & \epsilon & 6 \\ \epsilon & 4 & \epsilon \end{bmatrix} \oplus' \begin{bmatrix} 2 & 8 & 5 \\ 2 & 10 & 5 \\ 5 & \epsilon & 10 \end{bmatrix} \oplus' \begin{bmatrix} 3 & 9 & 6 \\ 3 & 9 & 6 \\ 6 & 14 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 8 & 4 \\ 1 & 9 & 5 \\ 5 & 4 & 9 \end{bmatrix}.$$

In the min-plus sum of these matrices, we have a catalogue of minimum-weight paths between the various nodes, or in the language of our application, the quickest lines of travel amongst the various cities. Something relatively novel has occurred in the above calculation. It turns out that the fastest means of travel between two cities is not always the path that links them directly. For instance, to go from city 3 to city 2 directly would take six units of time, while to travel to city 1 and then take the train from city 1 to city 2 would only require 5. Hence, $[A \oplus' A^{\otimes 2} \oplus' A^{\otimes 3}]_{23} = 5$.

While shortest path problems are themselves interesting, we introduce them in Chapter 1 mainly because the tropical sum of powers of a matrix will prove handy in our discussion of the existence and uniqueness of tropical eigenvalues. Indeed, the following theorem will be crucial in our proof of the Tropical Eigenvalue Theorem for irreducible matrices.

THEOREM 1.1. *Let $A \in \mathbb{R}_{\min}^{n\times n}$. If the average weight of any circuit in $G(A)$ is at least zero, then*

$$A^+ := \bigoplus_{k=1}^{\infty}{}' A^{\otimes k} = A \oplus' A^{\otimes 2} \oplus' \cdots \oplus' A^{\otimes n},$$

and $A^+ \in \mathbb{R}_{\min}^{n\times n}$.

PROOF. Let nodes $i, j \in V(G(A))$ be fixed but arbitrary. If there does not exist a path from j to i, then $[A^{\otimes k}]_{ij} = \epsilon$ for all $k \in \mathbb{N}$. Suppose there exists a path from j to i of a length greater than n; then this path is composed of at least one circuit and a path from j to i of a length that is at most n. Because circuits in $G(A)$ have nonnegative weights,

$$[A^+]_{ij} \geq \min\{[A^{\otimes k}]_{ij} : k \in \{1, 2, \ldots, n\}\},$$

which implies that the minimum path weight between nodes j and i is of a length less than or equal to n, as desired. □

5. Recurrence Relations

Many of the problems to which the theory of tropical eigenvectors and values can be applied involve modeled systems that change over time. One way of describing change in a system over time is with objects known as recurrence relations, or difference equations, as they

are commonly known. Recurrence relations do exactly what their name implies: they *relate* the current state of a model to its previous states, the latter of which *recur* in the formulation. If one prefers the alternate terminology, then difference equations can be said to show the *different* states of a modeled system as it changes over time. For an exploration of recurrence relations beyond tropical linear algebra, see Chapter 1 of [**4**].

In their most basic form, linear algebraic recurrence relations use some matrix A to describe the modeled system. Evolution of the system is described by

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

where $\mathbf{x}(k)$ represents the state of our system at time k. Thus, how the system evolves over time is entirely dependent upon the setup of our model in A and what we choose for our *initial condition* $\mathbf{x}(0)$.

As we shall see in the next chapter, we can model systems with tropical matrices and their states with tropical vectors. The recurrence relations with which we shall work take the form

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

Although the above exposition seems abstract, we will see in the next chapter just what types of systems these equations can describe and what exactly it means for these systems to be in a certain state.

CHAPTER 2

Unpacking the Tropical Eigenvalue-Vector Problem

We move now into our discussion of the tropical eigenvalue-vector problem. First, a motivation for solutions is presented. As it turns out, tropical eigenvalues and their associated eigenvectors can help us answer some interesting questions such as, how can we initiate a network of industrial assembly so that it produces in a stable, predictable fashion? Incorporating linear algebra, graph theory, and recurrence relations, discrete event systems can be thought of as the max-plus analogue to our minimum weight path problem. Section 2 will then show how the graph theoretic nature of tropical matrices can be used to prove the existence and uniqueness of tropical eigenvalues for irreducible matrices. A characterization of eigenspaces associated with these eigenvalues will, as a consequence of the proof, be given. Although this chapter's latter section works in the min-plus algebra, all results translate into the max-plus framework.

1. Discrete Event Systems as Motivation

We find in Chapter 1, Section 4 that weighted digraphs can be translated into min-plus matrices and vice-versa. Obviously, the same can be done for matrices in $\mathbb{R}_{\max}^{n\times n}$. In fact, matrices in the max-plus space and their associated graphs aptly describe a class of structures called discrete event dynamic systems (DEDS), more commonly referred to as

discrete event systems. Hence, in DEDS, we have a motivation for the tropical eigenvector-value problem. For a thorough overview of DEDS, see [**4**].

DEFINITION 2.1. A *discrete event dynamic system*, or DEDS, is a fabricated structure incorporating the resources and abilities of several entities for the purpose of achieving a common goal.

Take the manufacturing of a bicycle as an example of a discrete event system. The seat cannot be inserted into the frame until the frame itself is completely constructed. Hence, we see one entity, the stage at which we insert the seat, relying on another, the process of assembling the frame. Each stage is essential in working towards the overall goal: the assembly of the bicycle. Cuninghame-Green pioneered the use of the max-plus algebra and discrete event systems in describing industrial processes [**6**], [**7**].

We consider the graph of a discrete event system. Let $G = \{\{\eta_1, \eta_2, \eta_3, \eta_4, \eta_5\}, \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}, I_G, K_G\}$. Define I_G by $I_G(a_1) = (\eta_2, \eta_1)$, $I_G(a_2) = (\eta_3, \eta_2)$, $I_G(a_3) = (\eta_5, \eta_2)$, $I_G(a_4) = (\eta_1, \eta_3)$, $I_G(a_5) = (\eta_3, \eta_4)$, $I_G(a_6) = (\eta_5, \eta_4)$, and $I_G(a_7) = (\eta_4, \eta_5)$. Define K_G by $K_G(a_1) = 5$, $K_G(a_2) = 3$, $K_G(a_3) = 2$, $K_G(a_4) = 4$, $K_G(a_5) = 3$, $K_G(a_6) = 2$, and $K_G(a_7) = 1$.

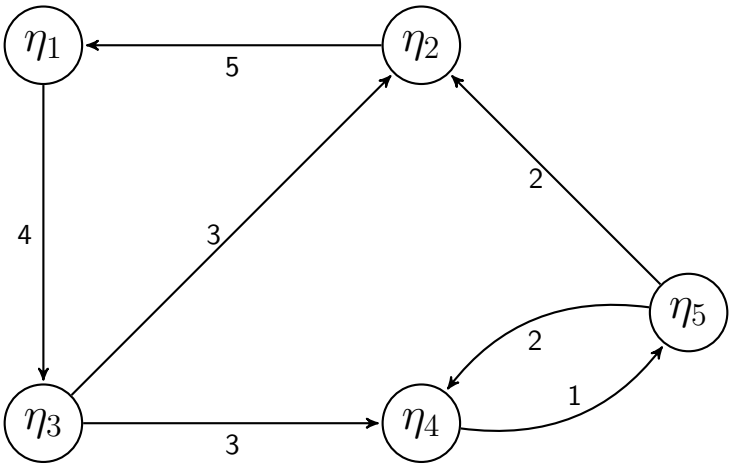


Fig. 2.1

Each node of our graph represents an entity in the system. The directed edges tell us how long it takes for a particular entity to communicate to another that it may begin its process. For instance, before η_1 can begin a cycle, it must wait to hear from η_2. This process of communication requires 5 time units. Note that in defining the mappings I_G and K_G we first listed all arcs that are directed towards node η_1, then η_2, and so on until η_5. This convention follows the matrix representation of directed graphs. Encoded in the matrix

$$A = \begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix}$$

is all of the information in the graph G, and we henceforth refer to G by $G(A)$. Note, moreover, that $A \in \mathbb{R}_{\max}^{5\times 5}$.

Of course, realistic systems will contain more than five entities, and the complexity of such networks gives rise to the problem of tracking

the behavior of individual nodes, whose behavior rely upon the processes of so many others, over multiple iterations of the system. Put another way, in order to study DEDS systematically, we must implement techniques that account for the common goal of the entities while tracking the behavior over time of each individual entity, as it is the former that relies upon the complex network of relationships between each of the latter.

Recurrence relations provide us with the language necessary for accomplishing these goals. Recall our expression of the standard tropical recurrence relation:

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

where $\mathbf{x}(k)$ represents the state of the system A at its k^{th} iteration, and $\mathbf{x}(k+1)$ relates the state one step later. In the context of discrete event systems, we say that the vector element $x_i(k)$ represents the earliest time at which node i can begin its k^{th} process. We will investigate this notion further by example.

Example 2.1. Consider the system defined by the graph in Figure 2.1. Recall $A \in \mathbb{R}_{\max}^{5\times 5}$. Suppose we choose as our initial condition for this system the vector $\mathbf{x}(0) = (1, \epsilon, 0, 1, 0)^\intercal$. To find when each node will begin its first cycle, we calculate $A \otimes \mathbf{x}(0) = \mathbf{x}(1)$, or

$$\begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \epsilon \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \epsilon \\ 3 \\ 5 \\ 3 \\ 2 \end{bmatrix}.$$

One aspect of the above computation worth noting is that $x_{\eta_1}(1) = \epsilon$. This means that η_1 cannot begin its process during the system's first cycle. To see why, we look at which nodes are "talking" to η_1. From figure 2.1, we see that node η_2 is the only vertex directly related to η_1. Thus, before η_1 can begin any process it must wait to hear from η_2. However, by establishing $x_{\eta_2}(0) = \epsilon$, we have, in effect, silenced node η_2 immediately prior to the first cycle. Furthermore, consider the fact that $x_{\eta_2}(1) = 3$. Looking at Figure 2.1, we find that nodes η_3 and η_5 are both talking to η_2. Since $x_{\eta_3}(0) = x_{\eta_5}(0) = 0$, the initial condition does not delay the activation of these two nodes, so they immediately begin their correspondences with node η_2. It takes 3 time units for η_3 to communicate to η_2 and 2 units for η_5. By the definition of tropical matrix multiplication, we choose the maximum of these communication times so that $x_{\eta_2}(1)$ represents the *earliest* time at which η_2 can begin its first process, as it must hear from both nodes before it can begin a cycle.

We have established a systematic technique for tracking discrete event systems as they evolve over time. However, this does us little good if we choose our initial condition such that the system evolves in an unstable, and even unpredictable manner. Indeed, to establish a dynamic system that behaves predictably, we may choose as its initial condition a tropical eigenvector of the corresponding matrix. The following example displays this point in action.

EXAMPLE 2.2. Working with the same system as in Example 2.1, we now fix $\mathbf{x}(0) = (5, 4, 5, 4, 1)^\intercal$. Therefore, $A \otimes \mathbf{x}(0) = \mathbf{x}(1)$ can be

represented as

$$\begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 5 \\ 4 \\ 5 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 8 \\ 9 \\ 8 \\ 5 \end{bmatrix}$$

$$= 4 \otimes \begin{bmatrix} 5 \\ 4 \\ 5 \\ 4 \\ 1 \end{bmatrix}.$$

Moreover, we have $A \otimes \mathbf{x}(1) = \mathbf{x}(2)$ computed as

$$\begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 9 \\ 8 \\ 9 \\ 8 \\ 5 \end{bmatrix} = \begin{bmatrix} 13 \\ 12 \\ 13 \\ 12 \\ 9 \end{bmatrix}$$

$$= 4 \otimes \begin{bmatrix} 9 \\ 8 \\ 9 \\ 8 \\ 5 \end{bmatrix}.$$

The behavior of the system in the above example relies on the scalar 4. In other words,

$$\begin{aligned} A \otimes \mathbf{x}(k) &= \mathbf{x}(k+1) \\ &= 4 \otimes \mathbf{x}(k). \end{aligned}$$

Thus, at any time k, node i can begin its k^{th} process 4 units after its $(k-1)$th process. In more general terms, when our system behaves such that

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

it moves forward in increments of λ. Therefore, to choose an ideal initial condition for a discrete event system, we find a tropical eigenvalue $\lambda \in \mathbb{R}$ and an associated eigenvector. We will develop algorithms for computing tropical eigenvalues and their associated eigenvectors, but first, we explore the abstract essence of the tropical eigenvalue from a min-plus perspective.

2. The Tropical Eigenvalue Theorem

We use the language of graph theory to develop a formal, theoretical understanding of the min-plus eigenvalue. For an exposition of the max-plus case, see [**2**] and [**7**], the former of which more closely resembles the following analysis. As it turns out, unique tropical eigenvalues exist for irreducible matrices. However, before we can show existence and uniqueness of tropical eigenvalues, we have a little more ground to cover. First, we give a formal definition of the tropical eigenvalue and eigenvector pair.

DEFINITION 2.2. Let $A \in \mathbb{R}_{\min}^{n\times n}$ ($\mathbb{R}_{\max}^{n\times n}$). If $\lambda \in \mathbb{R}_{\min}$ ($\mathbb{R}_{\max}$) and $\mathbf{v} \in \mathbb{R}_{\min}^{n}$ ($\mathbb{R}_{\max}^{n}$) such that $\mathbf{v}$ has at least one finite entry and

$$A \otimes \mathbf{v} = \lambda \otimes \mathbf{v},$$

then λ is an *eigenvalue* of A with associated eigenvector $\mathbf{v}$.

The first graph-theoretic feature of the tropical eigenvalue is articulated in the lemma, and its proof, below.

LEMMA 2.1. *Let $A \in \mathbb{R}_{\min}^{n\times n}$ have finite eigenvalue μ. Then a circuit γ exists in $G(A)$ such that*

$$\mu = \frac{|\gamma|_W}{|\gamma|_1}.$$

PROOF. Let $\mathbf{v}$ be an eigenvector associated with μ. By definition, $\mathbf{v}$ has at least one finite entry. There, thus, exists $\eta_1 \in V(G(A))$ such that $v_{\eta_1} \neq \epsilon$, and $[A \otimes \mathbf{v}]_{\eta_1}$ is equal to a finite value. By the way we define tropical matrix multiplication, this implies that there exists $\eta_2 \in V(G(A))$ such that

$$a_{\eta_1\eta_2} \otimes v_{\eta_2} = \mu \otimes v_{\eta_1},$$

further implying $a_{\eta_1\eta_2} \neq \epsilon$ and $v_{\eta_2} \neq \epsilon$. By the same logic, there exists $\eta_3 \in V(G(A))$ such that

$$a_{\eta_2\eta_3} \otimes v_{\eta_3} = \mu \otimes v_{\eta_2},$$

where $a_{\eta_2\eta_3} \neq \epsilon$ and $v_{\eta_3} \neq \epsilon$. Since $V(G(A))$ is finite, we will eventually find a repeat node, some node $\eta_h \in V(G(A))$ such that
$\eta_i = \eta_h$, with $i \in \{1, \ldots, h\}$. Thus, we have constructed a circuit γ of arbitrary length ℓ with $\gamma = ((\eta_h, \eta_{h+\ell-1}), (\eta_{h+\ell-1}, \eta_{h+\ell-2}), \ldots, (\eta_{h+1}, \eta_h))$.

Note $\eta_h = \eta_{h+\ell}$. Moreover,

$$|\gamma|_W = \bigotimes_{k=0}^{\ell-1} a_{\eta_{h+k}\eta_{h+k+1}}. \tag{2.1}$$

Since we have created γ on the basis that

$$a_{\eta_1\eta_2} \otimes v_{\eta_2} = \mu \otimes v_{\eta_1},$$

$$a_{\eta_2\eta_3} \otimes v_{\eta_3} = \mu \otimes v_{\eta_2},$$

and so on, we may write

$$\bigotimes_{k=0}^{\ell-1} \left(a_{\eta_{h+k}\eta_{h+k+1}} \otimes v_{\eta_{h+k+1}} \right) = \mu^{\otimes \ell} \otimes \bigotimes_{k=0}^{\ell-1} v_{\eta_{h+k}}.$$

We need to employ the standard algebraic operation of subtraction, so we express the above in classical terms:

$$\sum_{k=0}^{\ell-1} \left(a_{\eta_{h+k}\eta_{h+k+1}} + v_{\eta_{h+k+1}} \right) = \ell \times \mu + \sum_{k=0}^{\ell-1} v_{\eta_{h+k}}. \tag{2.2}$$

Note the following:

$$\sum_{k=0}^{\ell-1} v_{\eta_{h+k+1}} = v_{\eta_{h+1}} + v_{\eta_{h+2}} + \cdots + v_{\eta_{h+\ell+1}} + v_{\eta_{h+\ell}}$$

$$\sum_{k=0}^{\ell-1} v_{\eta_{h+k}} = v_{\eta_h} + v_{\eta_{h+1}} + \cdots + v_{\eta_{h+\ell-2}} + v_{\eta_{h+\ell+1}}.$$

Because $\eta_h = \eta_{h+\ell}$, we can see in the above expressions that

$$\sum_{k=0}^{\ell-1} v_{\eta_{h+k+1}} = \sum_{k=0}^{\ell-1} v_{\eta_{h+k}}.$$

Subtracting these two terms from (2.2), we have

$$\sum_{k=0}^{\ell-1} a_{\eta_{h+k}\eta_{h+k+1}} = \ell \times \mu,$$

and thus,

$$\bigotimes_{k=0}^{\ell-1} a_{\eta_{h+k}\eta_{h+k+1}} = \ell \times \mu.$$

From (2.1), we have

$$|\gamma|_W = \ell \times \mu,$$

and we conclude, since $|\gamma|_1 = \ell$,

$$\frac{|\gamma|_W}{|\gamma|_1} = \frac{1}{\ell} \times \ell \times \mu = \mu.$$

□

We know that tropical eigenvalues are average circuit weights in the corresponding graph. We can refine this statement further and claim that the *minimal* average circuit weight will always be an eigenvalue. Before we prove this, we define some necessary terms.

DEFINITION 2.3. Let $C(A)$ denote the set of all circuits of a graph $G(A)$. Fix

$$\lambda = \min_{p \in C(A)} \frac{|p|_W}{|p|_1}.$$

A circuit $\gamma \in C(A)$ is *critical* if $\lambda = \frac{|\gamma|_W}{|\gamma|_1}$. The *critical graph* of A, $G^C(A) = (V^C(G(A)), A^C(G(A)), I^C_{G(A)}, K^C_{G(A)})$, consists of all nodes and arcs that belong to critical circuits in $G(A)$.

To prove that minimal average circuit weights are eigenvalues, we also need the following lemma about critical graphs.

LEMMA 2.2. *Let $G(A)$ be a graph with at least one circuit. It follows that any circuit in $G^C(A)$ is critical.*

PROOF. We proceed with a proof by contradiction. Without loss of generality, assume that the scalar λ, as it is denoted in Definition 2.3, is

equal to 0. Suppose there exists a circuit ρ in $G^C(A)$ of arbitrary length $|\rho|_1 = \ell$. Assume $\frac{|\rho|_W}{|\rho|_1} < 0$. By definition, ρ is a circuit in $G(A)$, and we, therefore, have a contradiction, as this would make $\lambda = \frac{|\rho|_W}{|\rho|_1}$. Next, assume $\frac{|\rho|_W}{|\rho|_1} > 0$. Note that ρ can be represented as a concatenation of paths $\rho = \rho_1 \circ \rho_2 \circ \cdots \circ \rho_k$, and that each ρ_i is a subpath of a critical circuit c_i, $i \in \{1, 2, \ldots, k\}$. There then exist subpaths ξ_i such that $c_i = \xi_i \circ \rho_i$, $i \in \{1, 2, \ldots, k\}$. Since ρ is a circuit of length ℓ, we can order the c_i circuits as follows:

$c_1 = \xi_1 \circ \rho_1$ (ξ_1 begins at η_{h+1}, ends at η_h; ρ_1 begins at η_h, ends at η_{h+1})

$c_2 = \xi_2 \circ \rho_2$ (ξ_2 begins at η_{h+2}, ends at η_{h+1}; ρ_2 begins at η_{h+1}, ends at η_{h+2})

$\vdots$

$c_k = \xi_k \circ \rho_k$. (ξ_k begins at $\eta_{h+\ell}$, ends at $\eta_{h+\ell-1}$; ρ_k begins at $\eta_{h+\ell-1}$, ends at $\eta_{h+\ell}$)

It follows from our indexing of nodes that the concatenation $\xi_k \circ \xi_{k-1} \circ \cdots \circ \xi_1$ forms a circuit. For all $i \in \{1, 2, \ldots, k\}$, $\frac{|c_i|_W}{|c_i|_1} = 0$, and because $\frac{|\rho|_W}{|\rho|_1} > 0$, the circuit $\xi = \xi_k \circ \xi_{k-1} \circ \cdots \circ \xi_1$ has average weight less than 0, producing a contradiction identical to that which emerged when assuming $\frac{|\rho|_W}{|\rho|_1} < 0$. Since the average weights of circuits in $G^C(A)$ cannot be greater than or less than λ, we conclude they all have average weight λ, and are, therefore, critical. □

The next definition sets up our proof of minimal average circuit weights as eigenvalues.

DEFINITION 2.4. If we let $A \in \mathbb{R}_{\min}^{n \times n}$ and λ be defined as it is in Definition 2.3, then we can define a matrix A_λ as follows:

$$[A_\lambda]_{ij} = a_{ij} - \lambda.$$

We further define the matrix A_λ^+ by

$$A_\lambda^+ = A_\lambda \oplus' A_\lambda^{\otimes 2} \cdots \oplus' \cdots A_\lambda^{\otimes n}$$

Let A_λ^* be defined as follows:

$$A_\lambda^* = E \oplus' A_\lambda^+,$$

where E represents our $n \times n$ min-plus multiplicative identity matrix.

Finally, before we proceed with our claim, as a matter of notation, denote the k^{th} column of a matrix B by $[B]_{\cdot k}$.

LEMMA 2.3. *Let the graph $G(A)$ of matrix A have finite minimal average circuit weight λ. Then the scalar λ is an eigenvalue of A, and the column $[A_\lambda^*]_{\cdot\eta}$ is an eigenvector of A associated with λ for any node $\eta \in V^C(G(A))$.*

PROOF. First, note that the graphs of A and A_λ are identical, save their arc weights. Although the arc weights of $G(A)$ and $G(A_\lambda)$ differ, they vary uniformly by an amount of λ. In other words, for all circuits of $G(A)$ with average weight α, there exists a corresponding circuit in $G(A_\lambda)$ with average weight $\alpha - \lambda$. Then the minimal average circuit weight of $G(A_\lambda)$ is zero. Therefore, by Lemma 2.2, all circuits in $G^C(A_\lambda)$ have weight 0, and since every node of a critical graph is contained in a circuit, any path from a node in $G^C(A_\lambda)$ to itself has weight zero. Put another way, for all $\eta \in V^C(A)$,

$$[A_\lambda^+]_{\eta\eta} = 0. \tag{2.3}$$

Recall our notation for denoting column vectors of matrices. By Definition 2.4,

$$[A_\lambda^*]_{\cdot\eta} = [E \oplus' A_\lambda^+]_{\cdot\eta}.$$

For cases where $i \neq \eta$, the vector entry $[A^*_\lambda]_{i\eta} = \epsilon \oplus' [A^+_\lambda]_{i\eta}$, and when $i = \eta$, $[A^*_\lambda]_{i\eta} = 0 \oplus' [A^+_\lambda]_{i\eta}$. Therefore, from (2.3), we have

$$[A^+_\lambda]_{\cdot\eta} = [A^*_\lambda]_{\cdot\eta}, \tag{2.4}$$

for all $\eta \in N^C(A)$. Next, consider the following algebraic manipulations of our definition for A^+:

$$\begin{aligned}
A^+_\lambda &= A_\lambda \oplus' A^{\otimes 2}_\lambda \oplus' \cdots \oplus' A^{\otimes n}_\lambda \\
&= A_\lambda \oplus' A^{\otimes 2}_\lambda \oplus' \cdots \oplus' A^{\otimes n}_\lambda \oplus' A^{\otimes (n+1)}_\lambda \qquad \text{(by Theorem 1.1)} \\
&= A_\lambda \otimes (E \oplus' A_\lambda \oplus' A^{\otimes 2}_\lambda \oplus' \cdots \oplus' A^{\otimes n}_\lambda) \\
&= A_\lambda \otimes (E \oplus' A^+_\lambda) \\
&= A_\lambda \otimes A^*_\lambda.
\end{aligned}$$

We thus make the following substitution into (2.4):

$$[A_\lambda \otimes A^*_\lambda]_{\cdot\eta} = [A^*_\lambda]_{\cdot\eta},$$

and it follows

$$A_\lambda \otimes [A^*_\lambda]_{\cdot\eta} = [A^*_\lambda]_{\cdot\eta},$$

which, after "adding" λ (recall that $[A_\lambda]_{ij} = [A]_{ij} - \lambda$) to both sides of the above expression yields

$$A \otimes [A^*_\lambda]_{\cdot\eta} = \lambda \otimes [A^*_\lambda]_{\cdot\eta}.$$

Put into words, we have shown that for any node η in the critical graph of A, the η^{th} column of A_λ is an eigenvector of A associated with the eigenvalue λ. $\square$

Notice how the above lemma indicates that so long as the minimal average circuit weight of $G(A)$ is finite, there must exist an eigenvalue of A. By definition, if A is irreducible, then its minimum average circuit weight *will* be finite. Thus, if we narrow our gaze to irreducible matrices, we have the existence of eigenvalues. What follows is a proof of the fact that these eigenvalues are unique for irreducible tropical matrices.

Let $\gamma = ((\eta_1, \eta_2), (\eta_2, \eta_3), \ldots, (\eta_\ell, \eta_{\ell+1}))$ be an arbitrary circuit in the strongly connected graph $G(A)$, where $|\gamma|_1 = \ell$ and $\eta_{\ell+1} = \eta_1$. It follows that for all $k \in \{1, 2, \ldots, \ell\}$, $a_{\eta_{k+1}\eta_k} \neq \epsilon$. Moreover, suppose there exists a finite eigenvalue of A, μ. Let $\mathbf{v}$ be an eigenvector associated with μ. That $A \otimes \mathbf{v} = \mu \otimes \mathbf{v}$ implies $a_{\eta_{k+1}\eta_k} \otimes \mathbf{v}_{\eta_k} \geq \mu \otimes \mathbf{v}_{\eta_{k+1}}$. We can now set up an argument similar to that made in the proof of Lemma 2.1, whereby

$$\begin{aligned}
&\bigotimes_{k=0}^{\ell-1} \left(a_{\eta_{k+1}\eta_{k+2}} \otimes v_{\eta_{k+2}} \right) \geq \mu^{\otimes \ell} \otimes \bigotimes_{k=0}^{\ell-1} v_{\eta_{k+1}} \\
\implies &\sum_{k=0}^{\ell-1} \left(a_{\eta_{k+1}\eta_{k+2}} \otimes v_{\eta_{k+2}} \right) \geq \mu^{\otimes \ell} \otimes \sum_{k=0}^{\ell-1} v_{\eta_{k+1}} \\
&\sum_{k=0}^{\ell-1} (a_{\eta_{k+1}\eta_{k+2}}) \geq \ell \times \mu \\
&|\gamma|_W \geq \ell \times \mu \\
&\frac{|\gamma|_W}{|\gamma|_1} \geq \mu.
\end{aligned}$$

Since we chose γ arbitrarily, the above inequality must hold for the minimal average circuit weight of $G(A)$. Therefore, μ is less than or equal to the minimal average circuit weight of $G(A)$. However, from Lemma 2.1, we have that μ must be an average circuit weight, so λ is

the only finite eigenvalue of the irreducible matrix A.

Suppose now that ϵ is an eigenvalue of A associated with eigenvector $\mathbf{v}$. By definition, $\mathbf{v}$ has at least one finite entry that we shall call v_η. Since A is irreducible, and we can "get to" node η from any other node in $V(G(A))$, there exists a row α of A such that $a_{\alpha\eta}$ is finite. Hence, $\epsilon = [\epsilon \otimes \mathbf{v}]_\alpha = [A \otimes \mathbf{v}]_\alpha \leq a_{\alpha\eta} \otimes v_\eta$. However, $a_{\alpha\eta}$ and v_η are finite values, and we have a contradiction. Thus, ϵ cannot be an eigenvalue of an irreducible matrix, and we have the following theorem.

THE TROPICAL EIGENVALUE THEOREM FOR IRREDUCIBLE MATRICES. *Any irreducible matrix $A \in \mathbb{R}_{\min}^{n\times n}$ ($\mathbb{R}_{\max}^{n\times n}$) possesses one and only one eigenvalue. This eigenvalue, denoted $\lambda(A) \in \mathbb{R}_{\min}$ ($\mathbb{R}_{\max}$), is a finite number equal to the minimal (maximal) average weight of circuits in $G(A)$.*

The above theorem is worthy of being the culmination of our chapter. That an eigenvalue *must* exist for irreducible matrices, and that we have a graph theoretic name for this value will be invaluable in future computations. Next, we will look at the algorithms that dictate these computations.

CHAPTER 3

Computing Eigenvalues and Eigenvectors of Irreducible Matrices

Our main result in Chapter 2 is that any irreducible tropical matrix possesses a unique eigenvalue. We now concern ourselves with the computation of these eigenvalues. Two algorithms are presented: Karp's Algorithm and the Power Algorithm. Karp's will be given in the min-plus matrix algebra, but a max-plus analogue does exist; see [**2**]. Similarly, there is a min-plus version of the Power Algorithm, but our work will be restricted to the max-plus setting.

1. Karp's Algorithm

In 1978, Richard M. Karp devised an algorithm for computing eigenvalues of tropical matrices [**8**]. However, this was not the express purpose of Karp's paper, as he wrote, entirely in graph-theoretic terms, of computing minimum average circuit weights. We will, for now, concern ourselves strictly with its application to strongly connected graphs (graphs corresponding to irreducible matrices for which unique eigenvalues must exist). For an overview of the algorithm as it relates to tropical eigenvalues, see chapter 5, section 1 of [**3**], and for the max-plus case, one should also consult section 1 of chapter 5 in [**2**].

We will detail Karp's proof of his algorithm and work through a couple examples of its execution; we seek to illustrate, in a manner more detailed and instructive than currently exists in the literature,

how and why this method works. But first, we outline the algorithm.

KARP'S ALGORITHM. *Let $A \in \mathbb{R}_{\min}^{n\times n}$ be an irreducible matrix. To compute the eigenvalue of A, λ, we find the minimum average circuit weight in $G(A)$ by the following:*

(1) Choose an arbitrary $j \in V(G(A))$, and set $\mathbf{x}(0) = \mathbf{e}_j$, where $\mathbf{e}_j$ represents the jth column of the $n \times n$ tropical identity matrix E.

(2) Compute $\mathbf{x}(k) = A \otimes \mathbf{x}(k-1)$ for $k = 1, \ldots, n$.

(3) Compute

$$\lambda = \min_{i=1,\ldots,n} \left\{ \max_{k=0,\ldots,n-1} \left\{ \frac{x_i(n) - x_i(k)}{n-k} \right\} \right\}.$$

The proof of the algorithm requires the following lemma.

LEMMA 3.1. *Let $A \in \mathbb{R}_{\min}^{n\times n}$ be an irreducible matrix with eigenvalue $\lambda = 0$, $j \in \{1, \ldots, n\}$ be arbitrary, and $\mathbf{x}(k)$ denote $[A^{\otimes k}]_{\cdot j}$; then*

$$\min_{i=1,\ldots,n} \left\{ \max_{k=0,\ldots,n-1} \left\{ \frac{x_i(n) - x_i(k)}{n-k} \right\} \right\} = 0.$$

PROOF. Let $G(A)$ be the graph associated with A. Because $\lambda = 0$, the minimum average circuit weight of $G(A)$ is zero, and, thus, there exists a circuit γ in $G(A)$ such that $|\gamma|_W = 0$, and there cannot exist a circuit with negative weight. By Theorem 1.1, the minimal value of $[A^{\otimes k}]_{vj}$ is attained within n iterations of $A^{\otimes k}$. In graph theoretic terms, because $G(A)$ is strongly connected, there exists at least one path from node j to any node v, and if we let ρ denote the minimum-weight path from j to v; we have $|\rho|_1 \leq n$. Since $\min_{k=0,\ldots,n-1} x_v(k) = |\rho|_W$,

and $x_v(n) \geq |\rho|_W$,

$$x_v(n) - |\rho|_W = \max_{k=0,\ldots,n-1} \{x_v(n) - x_v(k)\} \geq 0,$$

and it follows from basic algebra that

$$\max_{k=0,\ldots,n-1} \left\{ \frac{x_v(n) - x_v(k)}{n-k} \right\} \geq 0.$$

If $x_v(n) = |\rho|_W$, then

$$\max_{k=0,\ldots,n-1} \left\{ \frac{x_v(n) - x_v(k)}{n-k} \right\} = 0,$$

and since there are no circuits with negative average weight, it would follow that

$$\min_{i=1,\ldots,n} \left\{ \max_{k=0,\ldots,n-1} \left\{ \frac{x_i(n) - x_i(k)}{n-k} \right\} \right\} = \max_{k=0,\ldots,n-1} \left\{ \frac{x_v(n) - x_v(k)}{n-k} \right\} = 0.$$

Thus, it suffices to show that there exists a node v such that $x_v(n) = |\rho|_W$. Recall $|\gamma|_W = 0$, and suppose that w is a node in γ. Let α represent the minimum weight path from j to w. Fix $|\alpha|_W = a$. If we concatenate any number of walks through γ to the path α, then we still have a minimum-weight progression of arcs from j to w. In other words, because $|\gamma|_W = 0$, for some concatenation of paths $\xi = \alpha \circ \gamma \circ \cdots \circ \gamma$, $|\xi|_W = a$. We will show, by contradiction, that this implies that any subpath of ξ, beginning at j and ending at some w' in γ, represents the minimum-weight path from j to w'. We provide Figure 3.1 to better illustrate the following argument. Note that the nodes x and y are included in γ to emphasize the fact that γ may traverse more nodes than w and w'. Let β represent a subpath of γ beginning at w and ending at w', and let ζ denote the path from w' to w over arcs in γ not

traversed by β. Fix $|\beta|_W = b$ and $|\zeta|_W = c$. Since $|\gamma|_W = 0$,

$$\begin{aligned} |\gamma|_W &= |\beta \circ \zeta|_W \\ &= |\beta|_W + |\zeta|_W \\ &= b + c \\ &= 0. \end{aligned}$$

But now suppose that there exists some minimum-weight path θ from j to w', not a subpath of ξ, such that $|\theta|_W = d$, and $d < a + b$. This implies that

$$\begin{aligned} d &< a + b \\ d + c &< a + b + c \\ d + c &< a \qquad\qquad \text{(recall } b + c = 0\text{)}. \end{aligned}$$

Note that $d + c$ represents the weight of the path $\theta \circ \zeta$ from j to w. However, we earlier assumed that α represents a minimum-weight path from j to w, so we cannot have a concatenation of paths distinct from α with weight less than a. Our construction of θ contradicts this assumption, and so we conclude that any subpath of ξ from j to w' is a minimum-weight path. Hence, let w' be the $(n+1)^{\text{th}}$ node of a subpath of ξ. Letting $v = w'$, we have shown the existence of a node v in $G(A)$ such that $x_v(n) = |\rho|_W$, and the proof of our lemma is complete. $\square$

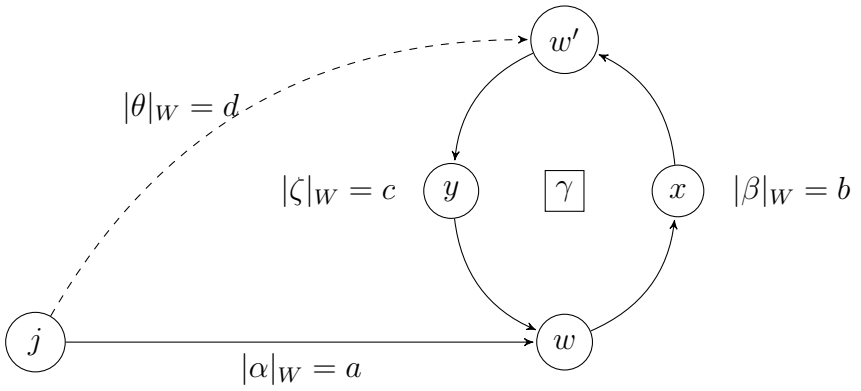


Figure 3.1

And now we may prove the validity of Karp's Algorithm in computing tropical eigenvalues.

THEOREM 3.2. *Let $A \in \mathbb{R}_{\min}^{n\times n}$ be an irreducible matrix with eigenvalue $\lambda \in \mathbb{R}_{\min}$, $j \in \{1,\dots,n\}$ be arbitrary, and $\mathbf{x}(k)$ denote $[A^{\otimes k}]_{\cdot j}$; then*

$$\lambda = \min_{i=1,\dots,n}\left\{\max_{k=0,\dots,n-1}\left\{\frac{x_i(n)-x_i(k)}{n-k}\right\}\right\}. \tag{1}$$

PROOF. Recall from Chapter 2 our definition of the matrix A_λ, that is, the matrix that results from subtracting the eigenvalue of A from each of its entries. Let A_c be similarly defined for some $c \in \mathbb{R}$. It is easily seen that the minimum average circuit weight of $G(A_c)$ is $\lambda - c$. Thus, the eigenvalue of A_c is $\lambda - c$. We seek to show that the expression

$$\min_{i=1,\dots,n}\left\{\max_{k=0,\dots,n-1}\left\{\frac{x_i(n)-x_i(k)}{n-k}\right\}\right\}$$

is also reduced by c. Let $\upsilon, r \in \{1,\dots,n\}$ such that

$$\begin{aligned}[A^{\otimes 2}]_{\upsilon j} &= (a_{\upsilon 1}\otimes a_{1j}) \oplus' \cdots \oplus' (a_{\upsilon r}\otimes a_{rj}) \oplus' \cdots \oplus' (a_{\upsilon n}\otimes a_{nj}) \\ &= a_{\upsilon r}\otimes a_{rj},\end{aligned}$$

and, since all entries in the matrix are evenly reduced,

$$\begin{aligned}[A_c^{\otimes 2}]_{vj} &= (a_{vr} - c) \otimes (a_{rj} - c) \\ &= a_{vr} \otimes a_{rj} - 2c.\end{aligned}$$

It follows, by induction on k, that

$$[A_c^{\otimes k}]_{vj} = [A^{\otimes k}]_{vj} - kc.$$

Hence,

$$\frac{x_v(n) - x_v(k)}{n-k}$$

from our A matrix, becomes

$$\begin{aligned}\frac{(x_v(n) - nc) - (x_v(k) - kc)}{n-k} &= \frac{x_v(n) - x_v(k) - nc + kc}{n-k} \\ &= \frac{x_v(n) - x_v(k) - c(n-k)}{n-k} \\ &= \frac{x_v(n) - x_v(k)}{n-k} - \frac{c(n-k)}{n-k} \\ &= \frac{x_v(n) - x_v(k)}{n-k} - c\end{aligned}$$

in our A_c matrix. Naturally,

$$\min_{i=1,\ldots,n}\left\{\max_{k=0,\ldots,n-1}\left\{\frac{x_i(n) - x_i(k)}{n-k} - c\right\}\right\} = \min_{i=1,\ldots,n}\left\{\max_{k=0,\ldots,n-1}\left\{\frac{x_i(n) - x_i(k)}{n-k}\right\}\right\} - c.$$

We could fix $c = \lambda$. By Lemma 3.1, the right side of (1) would, for the matrix A_λ, yield zero, the eigenvalue of A_λ. Translating back to A (adding λ to each entry of A_λ) would, as we have demonstrated above, add λ to both sides of (1). Hence,

$$\lambda = \min_{i=1,\ldots,n}\left\{\max_{k=0,\ldots,n-1}\left\{\frac{x_i(n) - x_i(k)}{n-k}\right\}\right\}.$$

□

The following example provides a straightforward exposition of the algorithm.

EXAMPLE 3.1. Let the irreducible matrix $A \in \mathbb{R}_{\min}^{n\times n}$ be defined as

$$A = \begin{bmatrix} 6 & 7 & \epsilon & \epsilon \\ 1 & 4 & \epsilon & 6 \\ \epsilon & 2 & 4 & 5 \\ \epsilon & \epsilon & 3 & 6 \end{bmatrix}.$$

We represent A graphically below.

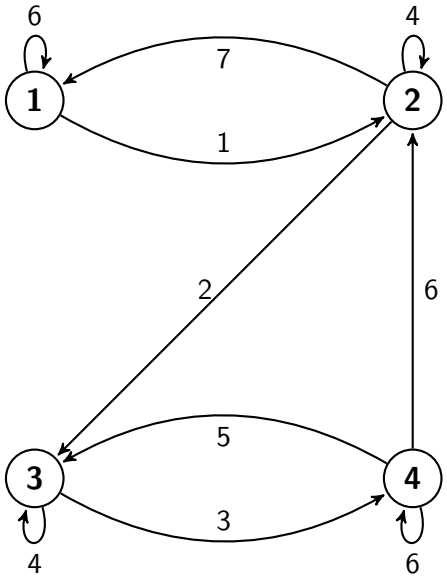


Figure 3.2

Since A is irreducible, we may use Karp's Algorithm to compute the tropical eigenvalue. Recall that we are in min-plus, so that $\epsilon = \infty$. In this computation, we shall let $j = 3$ so that $\mathbf{x}(0) = (\epsilon, \epsilon, 0, \epsilon)^{\intercal}$. Performing the second step of the algorithm, we find $\mathbf{x}(k)$ for $k =$

$1, 2, 3, 4$:

$$\mathbf{x}(1) = A \otimes \mathbf{x}(0) = \begin{bmatrix} 6 & 7 & \epsilon & \epsilon \\ 1 & 4 & \epsilon & 6 \\ \epsilon & 2 & 4 & 5 \\ \epsilon & \epsilon & 3 & 6 \end{bmatrix} \otimes \begin{bmatrix} \epsilon \\ \epsilon \\ 0 \\ \epsilon \end{bmatrix} = \begin{bmatrix} \epsilon \\ \epsilon \\ 4 \\ 3 \end{bmatrix};$$

$$\mathbf{x}(2) = A \otimes \mathbf{x}(1) = \begin{bmatrix} 6 & 7 & \epsilon & \epsilon \\ 1 & 4 & \epsilon & 6 \\ \epsilon & 2 & 4 & 5 \\ \epsilon & \epsilon & 3 & 6 \end{bmatrix} \otimes \begin{bmatrix} \epsilon \\ \epsilon \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} \epsilon \\ 9 \\ 8 \\ 7 \end{bmatrix};$$

$$\mathbf{x}(3) = A \otimes \mathbf{x}(2) = \begin{bmatrix} 6 & 7 & \epsilon & \epsilon \\ 1 & 4 & \epsilon & 6 \\ \epsilon & 2 & 4 & 5 \\ \epsilon & \epsilon & 3 & 6 \end{bmatrix} \otimes \begin{bmatrix} \epsilon \\ 9 \\ 8 \\ 7 \end{bmatrix} = \begin{bmatrix} 16 \\ 13 \\ 11 \\ 11 \end{bmatrix};$$

$$\mathbf{x}(4) = A \otimes \mathbf{x}(3) = \begin{bmatrix} 6 & 7 & \epsilon & \epsilon \\ 1 & 4 & \epsilon & 6 \\ \epsilon & 2 & 4 & 5 \\ \epsilon & \epsilon & 3 & 6 \end{bmatrix} \otimes \begin{bmatrix} 16 \\ 13 \\ 11 \\ 11 \end{bmatrix} = \begin{bmatrix} 20 \\ 17 \\ 15 \\ 14 \end{bmatrix}.$$

Recall our discussion of minimum-weight path problems in Chapter 1, where we have that the path of minimum weight of length k from node ℓ to node i in $G(A)$ is given by $[A^{\otimes k}]_{i\ell}$. In fact, in computing our $\mathbf{x}(k)$ vectors we have found $[A^{\otimes k}]_{\cdot j}$ for $k = 1, 2, 3, 4$, so that $\mathbf{x}(1), \mathbf{x}(2), \mathbf{x}(3), \text{and } \mathbf{x}(4)$ represent a catalogue of all minimum paths of lengths 1 through 4 beginning at node 3.

We next tackle the messy notation of step (3), a procedure by which,

using our catalogue of minimum path lengths, we isolate average circuit weights and choose the minimum of these values. Consider the difference quotients that we must maximize:

$$\max\left\{\frac{20-\epsilon}{4-0}, \frac{20-\epsilon}{4-1}, \frac{20-\epsilon}{4-2}, \frac{20-16}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{-\epsilon}{3}, \frac{-\epsilon}{2}, \frac{4}{1}\right\} = 4; \quad (i=1)$$

$$\max\left\{\frac{17-\epsilon}{4-0}, \frac{17-\epsilon}{4-1}, \frac{17-9}{4-2}, \frac{17-13}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{-\epsilon}{3}, \frac{6}{2}, \frac{4}{1}\right\} = 4; \quad (i=2)$$

$$\max\left\{\frac{15-0}{4-0}, \frac{15-4}{4-1}, \frac{15-8}{4-2}, \frac{15-11}{4-3}\right\} = \max\left\{\frac{15}{4}, \frac{11}{3}, \frac{7}{2}, \frac{4}{1}\right\} = 4; \quad (i=3)$$

$$\max\left\{\frac{14-\epsilon}{4-0}, \frac{14-3}{4-1}, \frac{14-7}{4-2}, \frac{14-11}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{11}{3}, \frac{7}{2}, \frac{3}{1}\right\} = \frac{11}{3}. \quad (i=4)$$

Finally, we minimize these maximums to find

$$\lambda = \min\left\{4, 4, 4, \frac{11}{3}\right\} = \frac{11}{3}.$$

Karp's Algorithm is perhaps best understood through an example resembling the matrix from our statement and proof of Lemma 3.1. Hence, we work through the algorithm again, this time computing the eigenvalue of A_λ, where $A \in \mathbb{R}_{\min}^{n\times n}$ is the matrix from our previous example.

EXAMPLE 3.2. We define $A_\lambda \in \mathbb{R}_{\min}^{n\times n}$ below and its graph $G(A_\lambda)$ in Figure 3.3:

$$A_\lambda = \begin{bmatrix} 2\frac{1}{3} & 3\frac{1}{3} & \epsilon & \epsilon \\ -2\frac{2}{3} & \frac{1}{3} & \epsilon & 2\frac{1}{3} \\ \epsilon & -1\frac{2}{3} & \frac{1}{3} & 1\frac{1}{3} \\ \epsilon & \epsilon & -\frac{2}{3} & 2\frac{1}{3} \end{bmatrix}.$$

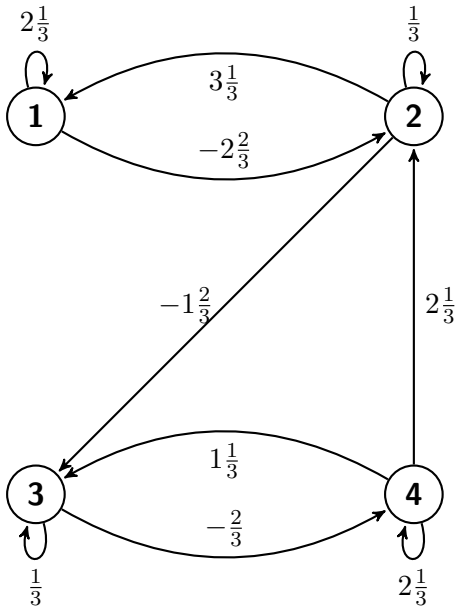


Figure 3.3

It is easily seen (see Theorem 3.2) that the eigenvalue of A_λ, λ^*, is equal to zero, but for the sake of exposition, we will detail the steps of Karp's Algorithm in computing λ^*. Fix $j = 1$, and hence, $\mathbf{x}(0) = (0, \epsilon, \epsilon, \epsilon)^\intercal$. For our sequence of $\mathbf{x}(k)$ vectors, we have

$$\mathbf{x}(1) = A \otimes \mathbf{x}(0) = \begin{bmatrix} 2\frac{1}{3} & 3\frac{1}{3} & \epsilon & \epsilon \\ -2\frac{2}{3} & \frac{1}{3} & \epsilon & 2\frac{1}{3} \\ \epsilon & -1\frac{2}{3} & \frac{1}{3} & 1\frac{1}{3} \\ \epsilon & \epsilon & -\frac{2}{3} & 2\frac{1}{3} \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \epsilon \\ \epsilon \\ \epsilon \end{bmatrix} = \begin{bmatrix} 2\frac{1}{3} \\ -2\frac{2}{3} \\ \epsilon \\ \epsilon \end{bmatrix};$$

$$\mathbf{x}(2) = A \otimes \mathbf{x}(1) = \begin{bmatrix} 2\frac{1}{3} & 3\frac{1}{3} & \epsilon & \epsilon \\ -2\frac{2}{3} & \frac{1}{3} & \epsilon & 2\frac{1}{3} \\ \epsilon & -1\frac{2}{3} & \frac{1}{3} & 1\frac{1}{3} \\ \epsilon & \epsilon & -\frac{2}{3} & 2\frac{1}{3} \end{bmatrix} \otimes \begin{bmatrix} 2\frac{1}{3} \\ -2\frac{2}{3} \\ \epsilon \\ \epsilon \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ -2\frac{1}{3} \\ -4\frac{1}{3} \\ \epsilon \end{bmatrix};$$

$$\mathbf{x}(3) = A \otimes \mathbf{x}(2) = \begin{bmatrix} 2\frac{1}{3} & 3\frac{1}{3} & \epsilon & \epsilon \\ -2\frac{2}{3} & \frac{1}{3} & \epsilon & 2\frac{1}{3} \\ \epsilon & -1\frac{2}{3} & \frac{1}{3} & 1\frac{1}{3} \\ \epsilon & \epsilon & -\frac{2}{3} & 2\frac{1}{3} \end{bmatrix} \otimes \begin{bmatrix} \frac{2}{3} \\ -2\frac{1}{3} \\ -4\frac{1}{3} \\ \epsilon \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ -4 \\ -5 \end{bmatrix};$$

$$\mathbf{x}(4) = A \otimes \mathbf{x}(3) = \begin{bmatrix} 2\frac{1}{3} & 3\frac{1}{3} & \epsilon & \epsilon \\ -2\frac{2}{3} & \frac{1}{3} & \epsilon & 2\frac{1}{3} \\ \epsilon & -1\frac{2}{3} & \frac{1}{3} & 1\frac{1}{3} \\ \epsilon & \epsilon & -\frac{2}{3} & 2\frac{1}{3} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ -2 \\ -4 \\ -5 \end{bmatrix} = \begin{bmatrix} 1\frac{1}{3} \\ -2\frac{2}{3} \\ -3\frac{2}{3} \\ -4\frac{2}{3} \end{bmatrix}.$$

Our difference quotients become

$$\max\left\{\frac{1\frac{1}{3}-0}{4-0}, \frac{1\frac{1}{3}-2\frac{1}{3}}{4-1}, \frac{1\frac{1}{3}-\frac{2}{3}}{4-2}, \frac{1\frac{1}{3}-1}{4-3}\right\} = \max\left\{\frac{\frac{4}{3}}{4}, \frac{-1}{3}, \frac{\frac{2}{3}}{2}, \frac{\frac{1}{3}}{1}\right\} = \frac{1}{3}; \qquad (i=1)$$

$$\max\left\{\frac{-2\frac{2}{3}-\epsilon}{4-0}, \frac{-2\frac{2}{3}+2\frac{2}{3}}{4-1}, \frac{-2\frac{2}{3}+2\frac{1}{3}}{4-2}, \frac{-2\frac{2}{3}+2}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{0}{3}, \frac{\frac{-1}{3}}{2}, \frac{\frac{-1}{3}}{1}\right\} = 0; \quad (i=2)$$

$$\max\left\{\frac{-3\frac{2}{3}-\epsilon}{4-0}, \frac{-3\frac{2}{3}-\epsilon}{4-1}, \frac{-3\frac{2}{3}+4\frac{1}{3}}{4-2}, \frac{-3\frac{2}{3}+4}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{-\epsilon}{3}, \frac{\frac{2}{3}}{2}, \frac{\frac{1}{3}}{1}\right\} = \frac{1}{3}; \qquad (i=3)$$

$$\max\left\{\frac{-4\frac{2}{3}-\epsilon}{4-0}, \frac{-4\frac{2}{3}-\epsilon}{4-1}, \frac{-4\frac{2}{3}-\epsilon}{4-2}, \frac{-4\frac{2}{3}+5}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{-\epsilon}{3}, \frac{-\epsilon}{2}, \frac{\frac{1}{3}}{1}\right\} = \frac{1}{3}, \qquad (i=4)$$

and we compute

$$\lambda^* = \min\left\{\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}\right\} = 0.$$

It is with this example, one for which the tropical eigenvalue of our matrix is zero, that we can relate the proof of Lemma 3.1 to the inner workings of Karp's Algorithm. Recall that we have set $j = 1$. Hence, because $\lambda^* = 0$, Lemma 3.1 guarantees the existence of a node $v \in V(G(A_\lambda))$ such that an n-length path from node 1 to v attains a minimum weight, and moreover, this path traverses a critical circuit of

$G(A_\lambda)$. In other words,

$$x_\upsilon(n) = \min_{k=0,\ldots,n-1} x_\upsilon(k).$$

Karp's Algorithm is designed to find this special node υ. To see how the Algorithm pinpoints υ from the catalogue of minimum-weight paths established by the $\mathbf{x}(k)$ vectors, let us note that in this example, $\upsilon = 2$. Consider the series of difference quotients when $i = 2$:

$$\max\left\{\frac{-2\frac{2}{3}-\epsilon}{4-0}, \frac{-2\frac{2}{3}+2\frac{2}{3}}{4-1}, \frac{-2\frac{2}{3}+2\frac{1}{3}}{4-2}, \frac{-2\frac{2}{3}+2}{4-3}\right\} = \max\left\{\frac{-\epsilon}{4}, \frac{0}{3}, \frac{\frac{-1}{3}}{2}, \frac{-\frac{1}{3}}{1}\right\} = 0.$$

The first term over which we maximize is infinite. In general, $\frac{\mathbf{x}_i(n)-\mathbf{x}_i(1)}{n}$ is only a candidate for the eigenvalue when the critical circuit is of length n. The second term considered is, of course, the eigenvalue 0. Node 2 is part of the zero-weight circuit described by $\gamma = (2,3)\circ(3,4)\circ(4,2)$, a circuit we picture below in the critical graph $G^C(A_\lambda)$.

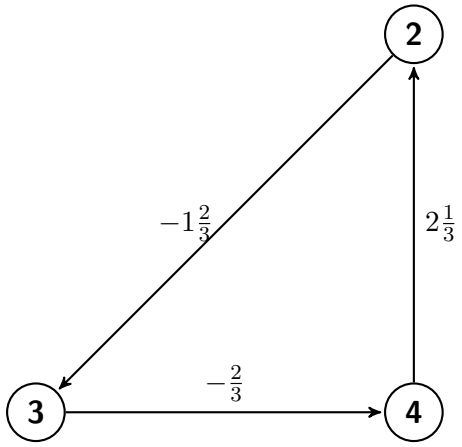


Figure 3.4

Furthermore, our calculations demonstrate $\upsilon = 2$ because they show that the minimum-weight arc progression from node 1 to node 2 is a 1-length path with weight $-2\frac{2}{3}$. That $|\gamma|_W = 0$ and $|\gamma|_1 = 3$ implies that the traversal of the 1-length minimum path from node 1 to node

2, followed by a trip around the critical circuit ending at node 2 gives us our 4-length path from 1 to 2 which attains the minimum weight. A graphical representation of this 4-length traversal also appears in Figure 3.5 below.

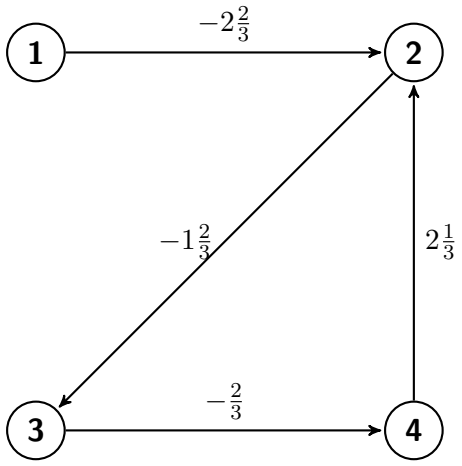


Figure 3.5

Hence, we use *difference* quotients to find the eigenvalue. Once the n-length path is found, we subtract off the weight of the excess path that brought us into the critical circuit. In this case, $|(2,3)|_1 = -2\frac{2}{3}$, so our difference is $-2\frac{2}{3} - (-2\frac{2}{3}) = 0$, and since we removed a 1-length path, we divide our difference by 3. That the subsequent candidates for maximization fail reveals why, despite the fact we are looking for the *minimum* cycle mean, we maximize these terms. With $\frac{-2\frac{2}{3}-(-2\frac{1}{3})}{4-2}$, we are removing a path that is not the one of minimum average weight from node 1 to 2. In seeking this minimum path, we want the *greatest* difference between $x_2(4)$ and any $x_2(k)$, and so we maximize the candidates. When removing a path that strays from our critical circuit, we will never, by Lemma 3.1, remove the shortest path from j to v.

Karp's Algorithm is not the most efficient method for computing tropical eigenvalues. Cataloguing every path of minimum weight

is a cumbersome, expensive process. The next algorithm we discuss provides a more efficient technique for solving the tropical eigenvalue-vector problem in the irreducible case.

2. The Power Algorithm

In Section 1 of Chapter 2, discrete event dynamic systems motivated our finding max-plus eigenvalues and associated eigenvectors. The discrete processes described by digraphs and tropical matrices can, in a sense, be controlled by solving the eigenvalue-vector problem. That is, the system described by $A \in \mathbb{R}_{\max}^{n\times n}$ will evolve predictably when we allow it to recur according to $\lambda \in \mathbb{R}_{\max}$ and $\mathbf{v} \in \mathbb{R}_{\max}^{n}$ such that we have $A \otimes \mathbf{v} = \lambda \otimes \mathbf{v}$. It turns out that for any irreducible tropical matrix, there is a point at which iterations of the form $A^{\otimes k}$ will themselves behave predictably, and the stability of these iterations is related to the matrix's tropical eigenvalues and eigenvectors [**2**]. The Power Algorithm makes use of this connection to compute tropical eigenvalues and eigenvectors of irreducible matrices more efficiently than Karp's.

THE POWER ALGORITHM. *Let $A \in \mathbb{R}_{\max}^{n\times n}$ be an irreducible matrix with eigenvalue λ and associated eigenvector $\mathbf{v}$. We find λ and $\mathbf{v}$ by the following:*

(1) Choose an arbitrary vector $\mathbf{x}(0) \in \mathbb{R}_{\max}^{n}$ such that $\mathbf{x}(0)$ has at least one finite entry.

(2) Iterate $A\otimes\mathbf{x}(k) = \mathbf{x}(k+1)$ (with $\mathbf{x}(0)$ as your initial condition) until a k is reached such that there exist $p \in \mathbb{N}$ and $c \in \mathbb{R}$, such that $\mathbf{x}(k+p) = c \otimes \mathbf{x}(k)$.

(3) Compute $\lambda = c/p$.

(4) Compute $\mathbf{v} = \bigoplus_{j=1}^{p} \left(\lambda^{\otimes(p-j)} \otimes \mathbf{x}\,(k+j-1)\right)$.

Before any meaningful discussion of the periodic behavior of $A^{\otimes k}$ iterations can occur, we require some more graph theory.

DEFINITION 3.1. Let G be a digraph with n nodes and $\eta_h, \eta_\ell \in V(G)$. We construct an equivalence relation $\sim_C$ on $V(G)$ by which $\eta_h \sim_C \eta_\ell$ if and only if there exists a circuit that traverses both η_h and η_ℓ. The equivalence classes under $\sim_C$ are defined by $V_i(G)$ (for $1 \leq i \leq q$), thus producing subgraphs G_i of G such that $E_i(G) = \{(\eta_h, \eta_\ell) \in E(G) : \eta_h, \eta_\ell \in V_i(G)\}$. Each G_i is called a *strongly connected component* of G. We define the *condensed graph* of $G(A)$, denoted $G(\tilde{A})$, by $V(G(\tilde{A})) = \{V_1(G(A)), V_2(G(A)), \ldots, V_q(G(A))\}$, and $(V_r(G(A)), V_s(G(A)) \in E(G(\tilde{A}))$ if and only if $r \neq s$, and there exists some $(\eta_h, \eta_\ell) \in E(G(A))$ such that $\eta_h \in V_r(G(A))$, and $\eta_\ell \in V_s(G(A))$.

DEFINITION 3.2. We define the *cyclicity* of a graph G, denoted σ_G, as follows:

- Let G be a strongly connected graph with n nodes. The cyclicity of G is equal to the greatest common divisor of the lengths of all circuits in G whose lengths are less than or equal to n.
- If G has one node and no directed arcs, then its cyclicity is one.
- If G is not strongly connected, then its cyclicity is equal to the least common multiple of the cyclicities of the strongly connected components of G.

DEFINITION 3.3. Let A be a matrix whose graph $G(A)$ contains at least one circuit. We define the *cyclicity of the matrix* A, denoted $\sigma(A)$, to be the cyclicity of the critical graph of A, $G^C(A)$.

We illuminate the above definitions by way of example.

EXAMPLE 3.3. Let $G(A)$ be the strongly connected graph pictured below in Figure 3.6 and described by

$$A = \begin{bmatrix} \epsilon & 3 & 4 \\ 2 & \epsilon & \epsilon \\ 4 & \epsilon & \epsilon \end{bmatrix}.$$

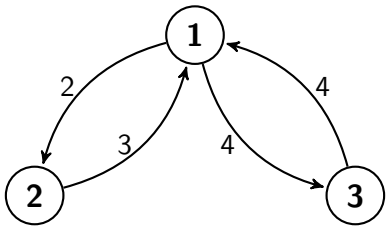


Figure 3.6

The graph $G(A)$ consists of two elementary circuits $\gamma_1 = (1,2) \circ (2,1)$ and $\gamma_2 = (1,3) \circ (3,1)$, and since it is strongly connected

$$\begin{aligned} \sigma_G &= \gcd\{|\gamma_1|_1, |\gamma_2|_1\} \\ &= \gcd\{2,2\} \\ &= 2. \end{aligned}$$

To find $\sigma(A)$, we consider the critical graph $G^C(A)$, which, because we are in $\mathbb{R}_{\max}^{n\times n}$ seeking the *maximal* average circuit weight, consists of nodes 1 and 3 and and the two 4-weight arcs creating a circuit between them. Then $\sigma(A) = \gcd\{|\gamma_2|_1\} = \gcd\{2\} = 2$. Notice what happens to the cyclicity of our graph when we add another arc, forming $G(A')$, so that

$$A' = \begin{bmatrix} 2 & 3 & 4 \\ 2 & \epsilon & \epsilon \\ 4 & \epsilon & \epsilon \end{bmatrix}.$$

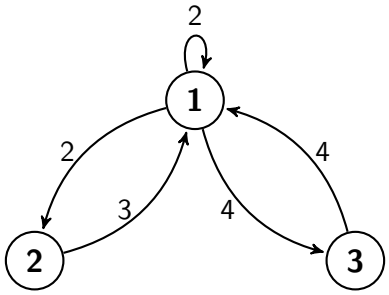


Figure 3.7

We have a new elementary circuit $\gamma_3 = (1, 1)$; hence,

$$\begin{aligned}\sigma_G &= \gcd\{|\gamma_1|_1, |\gamma_2|_1, |\gamma_3|_1\} \\ &= \gcd\{2, 2, 1\} \\ &= 1.\end{aligned}$$

However, our critical graph $G^C(A')$ remains unchanged. Therefore,

$$\sigma(A) = \sigma(A') = 2,$$

and we see that the cyclicities of graphs and their respective matrices need not exist in lockstep.

Example 3.4. Now we consider a graph that is not strongly connected. Let $G(A)$ be defined by the matrix

$$A = \begin{bmatrix} \epsilon & 2 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon & \epsilon \\ \epsilon & 5 & 6 & \epsilon \\ \epsilon & \epsilon & 5 & \epsilon \end{bmatrix}.$$

To determine σ_G, we first break up our graph into its strongly connected components. The partition of $V(G(A))$ under $\sim_C$ can be expressed as the collection $\{V_1(G(A)), V_2(G(A)), V_3(G(A))\} = \{\{1, 2\}, \{3\}, \{4\}\}$.

This partition helps us build the strongly connected components we illustrate in Figure 3.8:

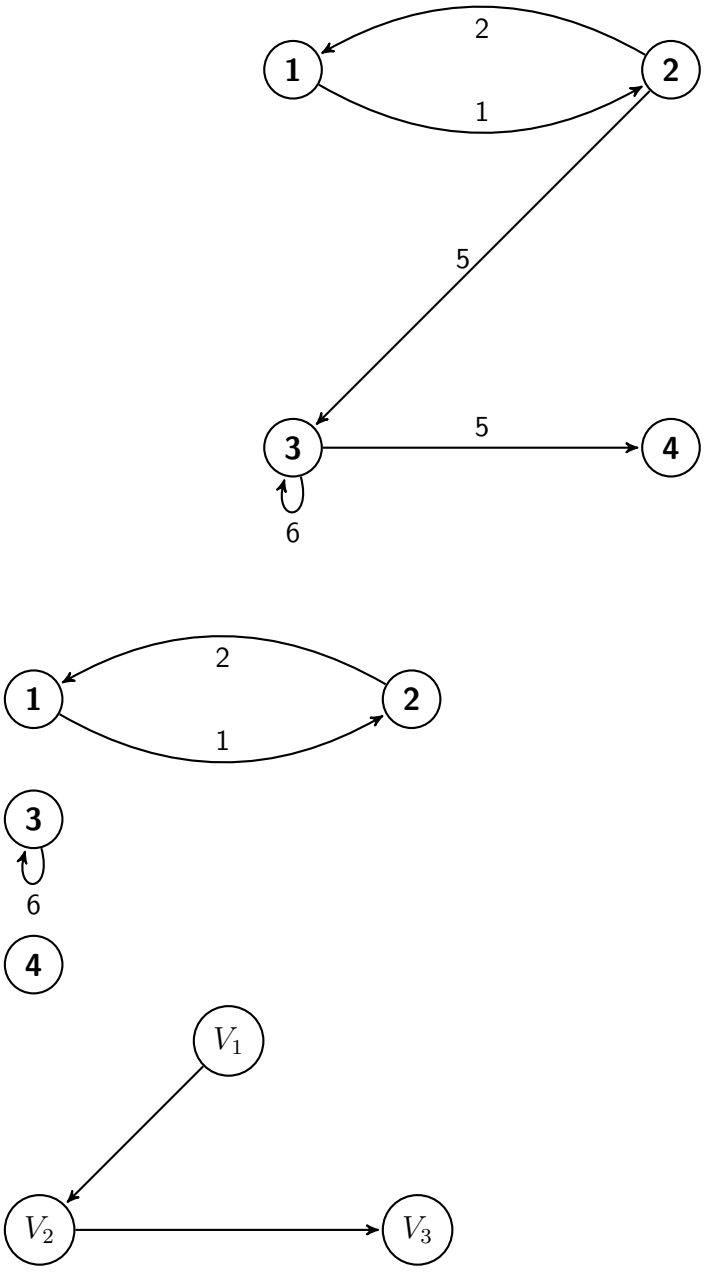


Figure 3.8

Thus,

$$\begin{aligned}\sigma_G &= \operatorname{lcm}\left\{\sigma_{G_1(A)}, \sigma_{G_2(A)}, \sigma_{G_3(A)}\right\} \\ &= \operatorname{lcm}\left\{\gcd\{2\}, \gcd\{1\}, 1\right\} \\ &= \operatorname{lcm}\{2, 1, 1\} \\ &= 2.\end{aligned}$$

The following theorem relates the cyclicity of a matrix to its natural powers, and it will be given without proof.

THEOREM 3.3. *Let $A \in \mathbb{R}_{\max}^{n\times n}$ be an irreducible matrix with eigenvalue λ and cyclicity $\sigma = \sigma(A)$. Then there exists $N \in \mathbb{N}$ such that*

$$A^{\otimes(k+\sigma)} = \lambda^{\otimes\sigma} \otimes A^{\otimes k}$$

for all $k \geq N$.

Theorem 3.3 implies periodic behavior from natural powers of irreducible matrices, and this behavior is contingent upon the cyclicity of these matrices. The following analysis, similar to [**1**], will further illustrate the relationship between cyclicity, eigenvalues, and eigenvectors of irreducible max-plus matrices.

Recall the matrix

$$A = \begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix}.$$

from Section 1 of Chapter 2. The following powers of A reveal aspects of its periodicity; we have

$$A^{\otimes 7} = \begin{bmatrix} 18 & 29 & 23 & 14 & 22 \\ 22 & 18 & 27 & 18 & 26 \\ 28 & 24 & 18 & 24 & 17 \\ 22 & 18 & 27 & 18 & 26 \\ 14 & 25 & 19 & 10 & 18 \end{bmatrix}; \quad A^{\otimes 8} = \begin{bmatrix} 27 & 23 & 32 & 23 & 31 \\ 31 & 27 & 21 & 27 & 20 \\ 22 & 33 & 27 & 18 & 26 \\ 31 & 27 & 21 & 27 & 20 \\ 23 & 19 & 28 & 19 & 27 \end{bmatrix};$$

$$A^{\otimes 9} = \begin{bmatrix} 36 & 32 & 26 & 32 & 25 \\ 25 & 36 & 30 & 21 & 29 \\ 31 & 27 & 36 & 27 & 35 \\ 25 & 36 & 30 & 21 & 29 \\ 32 & 28 & 22 & 28 & 21 \end{bmatrix}; \quad A^{\otimes 10} = \begin{bmatrix} 30 & 41 & 35 & 26 & 34 \\ 34 & 30 & 39 & 30 & 38 \\ 40 & 36 & 30 & 36 & 29 \\ 34 & 30 & 39 & 30 & 38 \\ 26 & 37 & 31 & 22 & 30 \end{bmatrix},$$

and, thus, $A^{\otimes 10} = 12 \otimes A^{\otimes 7}$. We found previously that the max-plus eigenvalue of A is $\lambda = 4$. In Figure 3.9, $G^C(A)$ is pictured, that is, the graph with all nodes and arcs from $G(A)$ contained in the circuit of average weight 4.

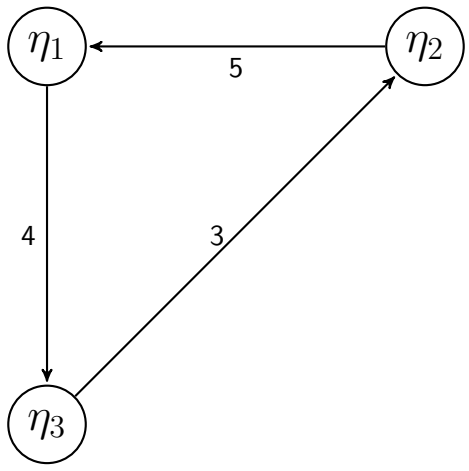


Fig. 3.9

The graph $G^C(A)$ only contains one circuit, and its length is 3, so $\sigma(A) = \gcd\{3\} = 3$. Given this information about λ, $\sigma(A)$, and powers of A, we conclude, by Theorem 3.3, that

$$A^{k+3} = 4^{\otimes 3} \otimes A^{\otimes k}$$

for all $k \geq 7$. Recalling the recurrence relation $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$, we have, when an initial condition $\mathbf{x}(0)$ is given and k is sufficiently large,

$$\begin{aligned}
\mathbf{x}(k+3) &= A^{\otimes(k+3)} \otimes \mathbf{x}(0) \\
&= (4^{\otimes 3} \otimes A^{\otimes k}) \otimes \mathbf{x}(0) \\
&= 4^{\otimes 3} \otimes (A^{\otimes k} \otimes \mathbf{x}(0)) \\
&= 4^{\otimes 3} \mathbf{x}(k) \\
&= 12 \otimes \mathbf{x}(k).
\end{aligned}$$

We will employ a general form of the above expressions to verify the Power Algorithm's computation of eigenvectors. Suppose that for natural numbers k and p and $c \in \mathbb{R}$, we have

$$\mathbf{x}(k+p) = c \otimes \mathbf{x}(k).$$

Fix $\lambda = \frac{c}{p}$, and let

$$\mathbf{v} = \mathbf{x}(k+p-1) \oplus (\lambda \otimes \mathbf{x}(k+p-2)) \oplus \cdots \oplus (\lambda^{\otimes(p-1)} \otimes \mathbf{x}(k)).$$

Then

$$\begin{aligned}
A \otimes \mathbf{v} &= (A \otimes \mathbf{x}(k+p-1)) \oplus (A \otimes \lambda \otimes \mathbf{x}(k+p-2)) \oplus \cdots \oplus (A \otimes \lambda^{\otimes(p-1)} \otimes \mathbf{x}(k)) \\
&= (A \otimes \mathbf{x}(k+p-1)) \oplus (\lambda \otimes A \otimes \mathbf{x}(k+p-2)) \oplus \cdots \oplus (\lambda^{\otimes(p-1)} \otimes A \otimes \mathbf{x}(k)) \\
&= \mathbf{x}(k+p) \oplus (\lambda \otimes \mathbf{x}(k+p-1)) \oplus \cdots \oplus (\lambda^{\otimes(p-1)} \otimes \mathbf{x}(k+1)) \\
&= (c \otimes \mathbf{x}(k)) \oplus (\lambda \otimes \mathbf{x}(k+p-1)) \oplus \cdots \oplus (\lambda^{\otimes(p-1)} \otimes \mathbf{x}(k+1)) \\
&= (\lambda^{\otimes p} \otimes \mathbf{x}(k)) \oplus (\lambda \otimes \mathbf{x}(k+p-1)) \oplus \cdots \oplus (\lambda^{\otimes(p-1)} \otimes \mathbf{x}(k+1)) \\
&= \lambda \otimes ((\lambda^{\otimes(p-1)} \otimes \mathbf{x}(k)) \oplus (\mathbf{x}(k+p-1)) \oplus \cdots \oplus (\lambda^{\otimes(p-2)} \otimes \mathbf{x}(k+1)) \\
&= \lambda \otimes (\mathbf{x}(k+p-1)) \oplus (\lambda \otimes \mathbf{x}(k+p-2)) \oplus \cdots \oplus (\lambda^{\otimes(p-1)} \otimes \mathbf{x}(k)) \\
&= \lambda \otimes \mathbf{v}.
\end{aligned}$$

Thus,

$$\mathbf{v} = \bigoplus_{j=1}^{p} \left(\lambda^{\otimes(p-j)} \otimes \mathbf{x}\left(k+j-1\right)\right),$$

as is stated in step (4) of the algorithm. The next example will run through the Power Algorithm using the above 5×5 matrix.

EXAMPLE 3.5. Since

$$A = \begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix}$$

is an irreducible tropical matrix, we can compute its eigenvalue λ and an associated eigenvector $\mathbf{v}$ using the Power Algorithm. First, we fix $\mathbf{x}(0) = \mathbf{e}_1 = (0, \epsilon, \epsilon, \epsilon, \epsilon)^\intercal$. Second, we iterate $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$ until

we uncover periodicity. This requires nine iterations:

$$\mathbf{x}(1) = A \otimes (0, \epsilon, \epsilon, \epsilon, \epsilon)^\intercal = (\epsilon, \epsilon, 4, \epsilon, \epsilon)^\intercal;$$

$$\mathbf{x}(2) = A \otimes (\epsilon, \epsilon, 4, \epsilon, \epsilon)^\intercal = (\epsilon, 7, \epsilon, 7, \epsilon)^\intercal;$$

$$\mathbf{x}(3) = A \otimes (\epsilon, 7, \epsilon, 7, \epsilon)^\intercal = (12, \epsilon, \epsilon, \epsilon, 8)^\intercal;$$

$$\mathbf{x}(4) = A \otimes (12, \epsilon, \epsilon, \epsilon, 8)^\intercal = (\epsilon, 10, 16, 10, \epsilon)^\intercal;$$

$$\mathbf{x}(5) = A \otimes (\epsilon, 10, 16, 10, \epsilon)^\intercal = (15, 19, \epsilon, 19, 11)^\intercal;$$

$$\mathbf{x}(6) = A \otimes (15, 19, \epsilon, 19, 11)^\intercal = (24, 13, 19, 13, 20)^\intercal;$$

$$\mathbf{x}(7) = A \otimes (24, 13, 19, 13, 20)^\intercal = (18, 22, 28, 22, 14)^\intercal;$$

$$\mathbf{x}(8) = A \otimes (18, 22, 28, 22, 14)^\intercal = (27, 31, 22, 31, 23)^\intercal;$$

$$\mathbf{x}(9) = A \otimes (27, 31, 22, 31, 23)^\intercal = (36, 35, 31, 25, 32)^\intercal;$$

Notice that $\mathbf{x}(9) = \mathbf{x}(6+3) = 12 \otimes \mathbf{x}(6)$, so after our ninth iteration in step (2), we may undertake step (3):

$$\lambda = \frac{12}{3} = 4.$$

Finally, we compute

$$\begin{aligned}
\mathbf{v} &= \bigoplus_{j=1}^{3} \left(4^{\otimes(3-j)} \otimes \mathbf{x}(6+j-1)\right) \\
&= 4^{\otimes 2} \otimes \mathbf{x}(6) \oplus 4^{\otimes 1} \otimes \mathbf{x}(7) \oplus 4^{\otimes 0} \otimes \mathbf{x}(8) \\
&= 8 \otimes (24, 13, 19, 13, 20)^\intercal \oplus 4 \otimes (18, 22, 28, 22, 14)^\intercal \oplus 0 \otimes (27, 31, 22, 31, 23)^\intercal \\
&= (32, 21, 27, 21, 28)^\intercal \oplus (22, 26, 32, 26, 18)^\intercal \oplus (27, 31, 22, 31, 23)^\intercal \\
&= (32, 31, 32, 31, 28)^\intercal,
\end{aligned}$$

and find that, in fact,

$$\begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 32 \\ 31 \\ 32 \\ 31 \\ 28 \end{bmatrix} = \begin{bmatrix} 36 \\ 35 \\ 36 \\ 35 \\ 32 \end{bmatrix}$$

$$= 4 \otimes \begin{bmatrix} 32 \\ 31 \\ 32 \\ 31 \\ 28 \end{bmatrix}.$$

EXAMPLE 3.6. For this example, the above matrix $A \in \mathbb{R}_{\max}^{5\times 5}$ and its corresponding graph in Figure 2.1 describe a discrete event system. Suppose the information encoded in A and $G(A)$ abstracts the workings of a radio communication network. Each node represents a tower; a weighted arc running from node j to node i (also denoted a_{ij}) gives the time it takes, in seconds, for tower j to construct and send a message to tower i and for tower i to receive and process the communication. Suppose, furthermore, that the network is automated so that its operator can program each tower with a start time and the towers will then carry out subsequent processes on their own accord. How can we set the network "into motion" so that, without having to iterate $A \otimes \mathbf{x}(k)$ for $k = 0, 1, \ldots, 6$, we know exactly how long it will take to run five times? The answer lies in our solution from Example 3.5. We found that $\lambda = 4$ is an eigenvalue of A with associated eigenvector $\mathbf{v} = (32, 31, 32, 31, 28)^{\intercal}$. Were we to set $\mathbf{v} = \mathbf{x}(0)$, the system would behave predictably, according to λ, and we would know exactly when

the fifth process will end. However, this would be impractical, as delaying communications for 28 to 32 seconds would be a waste of time. To derive a more practical solution, we simply find the least multiple of 4 in $\mathbf{v}$ and subtract it from each element, or tropically speaking,

$$-28 \otimes (32, 31, 32, 31, 28)^{\intercal} = (4, 3, 4, 3, 0)^{\intercal}.$$

Since we reduced $\mathbf{v}$ by a multiple of 4, the vector $(4, 3, 4, 3, 0)^{\intercal}$ is still in the eigenspace associated with λ. We verify this below:

$$\begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ 4 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 3 & \epsilon & 2 \\ \epsilon & \epsilon & \epsilon & 1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 4 \\ 3 \\ 4 \\ 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 8 \\ 7 \\ 4 \end{bmatrix}$$

$$= 4 \otimes \begin{bmatrix} 4 \\ 3 \\ 4 \\ 3 \\ 0 \end{bmatrix}.$$

Setting $\mathbf{x}(0) = (4, 3, 4, 3, 0)^{\intercal}$, gives us both predictable behavior and an efficient means of operating the network. Tower 5 will begin communicating immediately, while all other towers will be delayed by either 4 or 3 seconds. Note that it follows that

$$\begin{aligned} A \otimes \mathbf{x}(5) &= \mathbf{x}(6) \\ &= A^{\otimes 5} \otimes \mathbf{x}(0) \\ &= (28, 27, 28, 27, 24)^{\intercal}. \end{aligned}$$

The greatest value in $\mathbf{x}(6)$, 28, tells us that the sixth process of nodes in the system can begin 28 seconds after commencement. This reveals that the fifth process will be executed in 28 seconds, and we have, thus, solved the problem posed above.

In the Power Algorithm, we have a method for computing eigenvalues of irreducible tropical matrices, one more efficient than Karp's. Like with Karp's Algorithm, we are not guaranteed a solution to the eigenvalue-vector problem if our matrix is reducible. If a reducible matrix does not possess an eigenvalue, then the Power Algorithm will not terminate. One particularly interesting feature of Karp's Algorithm is that, regardless of whether its input possesses an eigenvalue, it will always terminate. Our next chapter investigates the consequences of this aspect of Karp's Algorithm, and whether or not we can learn anything from running it on matrices without eigenvalues.

CHAPTER 4

Periodicity of Reducible Matrices

Even though a reducible matrix A need not possess a tropical eigenvalue, iterations of the form

$$A \otimes \mathbf{x}(k) = \mathbf{x}(k+1), \text{ or}$$
$$A^{\otimes k} \otimes \mathbf{x}(0) = \mathbf{x}(k). \qquad (1)$$

still possess stable long-term behavior, so long as A contains finite entries in each row. Matrices with at least one finite entry in every row are called *regular*. This aforementioned stability is described by a pair of finite vectors called the generalized eigenmode. Section 1 introduces the theory underlying the eigenmode. Our analysis is restricted to $\mathbb{R}_{\max}^{n\times n}$. Indeed, work on the periodicity of matrices in $\mathbb{R}_{\min}^{n\times n}$ remains absent from the literature. In Section 2, we present Howard's Algorithm, a method for computing the generalized eigenmode of reducible systems. We explore the implementation of Karp's Algorithm in the reducible case in Section 3 and we save the fourth and final section for concluding remarks.

1. The Cycle-Time Vector and the Generalized Eigenmode

Shifting our perspective slightly, consider the $\mathbf{x}(k)$ vectors in iterations like (1) to be elements in the range of a sequence mapping $k = 0, 1, 2, \ldots$ to $\mathbb{R}_{\max}^{n}$. Much of the following analysis looks at the limit of this sequence as $k \to \infty$ (see [**2**]).

DEFINITION 4.1. Let $\{\mathbf{x}(k) : k \in \mathbb{N}\}$ be a sequence in $\mathbb{R}^n_{\max}$. If for all $j \in \{1, 2, \ldots, n\}$ the quantity

$$\eta_j = \lim_{k\to\infty} \frac{x_j(k)}{k}$$

exists, then the vector $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_n)^\intercal$ is the *cycle-time vector* of the sequence. If all elements of $\boldsymbol{\eta}$ have the same value, then this scalar is called the *asymptotic growth rate* of the sequence.

We can show that, assuming existence, the cycle-time vector of a sequence in $\mathbb{R}^n$ is unique. Essential to our proof is the following definition.

DEFINITION 4.2. We define the *l^∞-norm* of a vector $\mathbf{v} \in \mathbb{R}^n_{\max}$ to be the entry in $\mathbf{v}$ with maximal absolute value. Hence,

$$||\mathbf{v}||_\infty = \max_{i=1,\ldots,n} |v_i|.$$

Note that the l^∞-norm of a vector in $\mathbb{R}^n_{\max}$ with at least one entry equal to ϵ will be infinite. Thus, the following analysis deals with wholly finite vectors in proving uniqueness of the cycle-time vector.

LEMMA 4.1. *Let $A \in \mathbb{R}^{m\times n}_{\max}$ be a regular matrix (note A need not be square). Then*

$$||(A \otimes \mathbf{u}) - (A \otimes \mathbf{v})||_\infty \leq ||\mathbf{u} - \mathbf{v}||_\infty,$$

for any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$.

PROOF. Since A is a regular matrix, having at least one finite entry in each row, and $\mathbf{u}$ and $\mathbf{v}$ are themselves finite, we know $A \otimes \mathbf{u}, A \otimes \mathbf{v} \in \mathbb{R}^m$. We can, therefore, fix some $\alpha \in \mathbb{R}$ such that

$$\alpha = ||(A \otimes \mathbf{u}) - (A \otimes \mathbf{v})||_\infty,$$

and there exists $i_0 \in \{1, 2, \ldots, m\}$ such that

$$\alpha = \left| [(A \otimes \mathbf{u}) - (A \otimes \mathbf{v})]_{i_0} \right|.$$

If we assume $[(A \otimes \mathbf{u}) - (A \otimes \mathbf{v})]_{i_0} \geq 0$, then we have

$$\alpha = \max_{j=1,\ldots,n} (a_{i_0 j} + u_j) - \max_{l=1,\ldots,n} (a_{i_0 l} + v_l). \tag{2}$$

Let $j_0 \in \{1, 2, \ldots, n\}$ be the value that corresponds to the maximum attained in the first term of (2). Hence,

$$\alpha = (a_{i_0 j_0} + u_{j_0}) - \max_{l=1,\ldots,n} (a_{i_0 l} + v_l).$$

Obviously,

$$(a_{i_0 j_0} + v_{j_0}) \leq \max_{l=1,\ldots,n} (a_{i_0 l} + v_l),$$

so

$$\begin{aligned} \alpha &= (a_{i_0 j_0} + u_{j_0}) - \max_{l=1,\ldots,n} (a_{i_0 l} + v_l) \\ &\leq (a_{i_0 j_0} + u_{j_0}) - (a_{i_0 j_0} + v_{j_0}). \end{aligned}$$

Note that

$$(a_{i_0 j_0} + u_{j_0}) - (a_{i_0 j_0} + v_{j_0}) = u_{j_0} - v_{j_0}.$$

We have, then, that if $\alpha \geq 0$,

$$\begin{aligned} \alpha &\leq u_{j_0} - v_{j_0} \\ &\leq \max_{j=0,\ldots,n} (u_j - v_j) \\ &\leq \max_{j=0,\ldots,n} |u_j - v_j| \\ &\leq ||u_j - v_j||_\infty. \end{aligned}$$

To show the case for $\alpha \leq 0$, we need only swap the terms of (2) and perform the same analysis. Thus, we conclude that for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and regular $A \in \mathbb{R}^{m\times n}_{\max}$, $||(A \otimes \mathbf{u}) - (A \otimes \mathbf{v})||_\infty \leq ||\mathbf{u} - \mathbf{v}||_\infty$. □

Since the above proof works with an arbitrary regular matrix, and powers of matrices with finite entries in every row will also have finite entries in every row, it follows that for any $k \geq 0$

$$||(A^{\otimes k} \otimes \mathbf{u}) - (A^{\otimes k} \otimes \mathbf{v})||_\infty \leq ||\mathbf{u} - \mathbf{v}||_\infty. \tag{3}$$

Hence, the l^∞-distance between any two vectors from distinct sequences $(\mathbf{x}(k))_{k=0}^\infty$ and $(\mathbf{y}(k))_{k=0}^\infty$, where $\mathbf{x}(0) = \mathbf{u}$ and $\mathbf{y}(0) = \mathbf{v}$ is bounded by $||\mathbf{u} - \mathbf{v}||_\infty$. We employ this property of the l^∞-norm, known as non-expansiveness, to show that if a cycle-time vector of a given sequence exists, then it is unique and exists for any finite initial condition.

THEOREM 4.2. *Let $A \in \mathbb{R}^{n\times n}_{\max}$ be a square regular matrix, and consider the the sequence of vectors defined by $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$ with initial condition $\mathbf{x}(0) \in \mathbb{R}^n$. If a cycle-time vector exists for this sequence, that is, there is some $\boldsymbol{\eta} \in \mathbb{R}^n$ such that*

$$\lim_{k\to\infty} \frac{A^{\otimes k} \otimes \mathbf{x}(0)}{k} = \boldsymbol{\eta},$$

then this limit exists and has the same value for any initial condition $\mathbf{y}(0) \in \mathbb{R}^n$.

PROOF. By definition,

$$\begin{aligned} 0 &\leq \left|\left| \frac{A^{\otimes k} \otimes \mathbf{y}(0)}{k} - \frac{A^{\otimes k} \otimes \mathbf{x}(0)}{k} \right|\right|_\infty \\ &\leq \frac{1}{k} \left|\left| A^{\otimes k} \otimes \mathbf{y}(0) - A^{\otimes k} \otimes \mathbf{x}(0) \right|\right|_\infty, \end{aligned}$$

and from Lemma 4.1, we have that

$$\frac{1}{k}\left|\left|A^{\otimes k}\otimes\mathbf{y}(0)-A^{\otimes k}\otimes\mathbf{x}(0)\right|\right|_{\infty}\leq\frac{1}{k}\left|\left|\mathbf{y}(0)-\mathbf{x}(0)\right|\right|_{\infty},$$

which implies

$$\lim_{k\to\infty}\frac{1}{k}\left|\left|\mathbf{y}(0)-\mathbf{x}(0)\right|\right|_{\infty}\geq\lim_{k\to\infty}\left|\left|\frac{A^{\otimes k}\otimes\mathbf{y}(0)}{k}-\frac{A^{\otimes k}\otimes\mathbf{x}(0)}{k}\right|\right|_{\infty}\geq\lim_{k\to\infty}0$$

$$0\geq\lim_{k\to\infty}\left|\left|\frac{A^{\otimes k}\otimes\mathbf{y}(0)}{k}-\frac{A^{\otimes k}\otimes\mathbf{x}(0)}{k}\right|\right|_{\infty}\geq 0.$$

Thus,

$$\lim_{k\to\infty}\left|\left|\frac{A^{\otimes k}\otimes\mathbf{y}(0)}{k}-\frac{A^{\otimes k}\otimes\mathbf{x}(0)}{k}\right|\right|_{\infty}=0,$$

or qualitatively speaking, the greatest difference between corresponding entries of vectors in $(\mathbf{x}(k))_{k=0}^{\infty}$ and $(\mathbf{y}(k))_{k=0}^{\infty}$ divided by k tends to zero as $k\to\infty$. Hence, that

$$\lim_{k\to\infty}\frac{A^{\otimes k}\otimes\mathbf{x}(0)}{k}=\boldsymbol{\eta}$$

implies

$$\lim_{k\to\infty}\frac{A^{\otimes k}\otimes\mathbf{y}(0)}{k}=\boldsymbol{\eta}.$$

□

The language of sequences was useful in proving the uniqueness of the cycle-time vector of a recurrence relation $A^{\otimes k}\otimes\mathbf{x}(0)=\mathbf{x}(k)$. However, now that we have shown that the value $\boldsymbol{\eta}\in\mathbb{R}^n$ is independent of the initial condition $\mathbf{x}(0)\in\mathbb{R}^n$, we can talk about cycle-time vectors belonging to *matrices*, not just the particular sequences that reveal their value. Moreover, the cycle-time vector of a regular matrix $A\in\mathbb{R}_{\max}^{n\times n}$ is a generalized extension of the tropical eigenvalue, so that given a tropical matrix A with eigenvalue $\lambda\in\mathbb{R}$ and associated eigenvector $\mathbf{v}\in\mathbb{R}^n$, its cycle-time vector is $\boldsymbol{\eta}=(\lambda,\lambda,\ldots,\lambda)^{\intercal}\in\mathbb{R}^n$.

Note that reducible matrices may have multiple eigenvalues. However, since we have shown the cycle-time vector corresponding to sequences of wholly finite vectors (vectors in $\mathbb{R}^n$) to be unique, it follows that all regular tropical matrices have only one eigenvalue with finite associated eigenvectors.

THEOREM 4.3. *If $\mathbf{x}(k+1) = A \otimes \mathbf{x}(k)$ is a recurrence relation with the irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ having the eigenvalue $\lambda \in \mathbb{R}$, then for all $j \in \{1, 2, \ldots, n\}$,*

$$\lim_{k\to\infty} \frac{x_j(k)}{k} = \lambda$$

for any initial condition $\mathbf{x}(0) \in \mathbb{R}^n$.

PROOF. Let $\mathbf{v}$ be an eigenvector associated with λ. Note that because A is irreducible, $\mathbf{v} \in \mathbb{R}^n$. If we fix $\mathbf{x}(0) = \mathbf{v}$,

$$\begin{aligned}\mathbf{x}(k) &= A^{\otimes k} \otimes \mathbf{x}(0)\\ &= A^{\otimes k} \otimes \mathbf{v}\\ &= \lambda^{\otimes k} \otimes \mathbf{v}\end{aligned}$$

for all $k \geq 0$. Thus, by an abuse of notation ($+$ and $\times$ denote standard addition and multiplication), we have

$$\begin{aligned}\lim_{k\to\infty} \frac{x_j(k)}{k} &= \lim_{k\to\infty} \frac{\lambda^{\otimes k} \otimes v_j}{k}\\ &= \lim_{k\to\infty} \frac{k \times \lambda}{k} + \lim_{k\to\infty} \frac{v_j}{k}\\ &= \lambda + 0\\ &= \lambda,\end{aligned}$$

for all $j \in \{1, 2, \ldots, n\}$. By Theorem 4.2, the cycle-time vector of A is independent of the initial condition $\mathbf{x}(0)$, so we have proven the above theorem. □

The above analysis explains why we can choose an arbitrary vector when performing the Power Algorithm. For an irreducible matrix, the cycle-time vector, and, thus, the eigenvalue will reveal itself in iterations of $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$, regardless of the value of $\mathbf{x}(0)$.

We stated earlier that the cycle-time vector extends the notion of periodicity to matrices that may not necessarily possess an eigenvalue. Our next definition will complete the picture for regular tropical matrices by generalizing the eigenvalue-vector pair.

Definition 4.3. Let $+$ and $\times$ denote standard addition and multiplication, respectively. The pair of vectors $(\boldsymbol{\eta}, \mathbf{v}) \in \mathbb{R}^n \times \mathbb{R}^n$ is called a *generalized eigenmode* of the regular matrix $A \in \mathbb{R}_{\max}^{n \times n}$ if for all $k \geq 0$,

$$A \otimes (k \times \boldsymbol{\eta} + \mathbf{v}) = (k+1) \times \boldsymbol{\eta} + \mathbf{v}.$$

Assume that $(\boldsymbol{\eta}, \mathbf{v})$ constitute a generalized eigenmode for a regular matrix A, and let $\mathbf{x}(0) = \mathbf{v}$ in a recurrence relation. By Definition 4.3,

$$\begin{aligned} \mathbf{x}(k) &= k \times \boldsymbol{\eta} + \mathbf{v} \\ &= \boldsymbol{\eta}^{\otimes k} \otimes \mathbf{v} \end{aligned}$$

for all $k \geq 0$. Thus, $\lim_{k \to \infty} \dfrac{x_j(k)}{k} = \eta_j$ for all $j \in \{1, 2, \ldots, n\}$, and the cycle-time vector of A, which is uniquely determined, coincides with the vector $\boldsymbol{\eta}$ in the generalized eigenmode. Therefore, just as the eigenvalue and associated eigenspace of an irreducible tropical matrix is unique, so too is the vector $\boldsymbol{\eta}$ in the generalized eigenmode of a regular matrix (given that it exists). The vector $\mathbf{v}$ is not, however, uniquely

determined. In fact, given a generalized eigenmode $(\boldsymbol{\eta}, \mathbf{v})$ of A, the pair $(\boldsymbol{\eta}, c \otimes \mathbf{v})$ also constitutes an eigenmode of A, where $c \in \mathbb{R}$.

We verify the existence of a generalized eigenmode of any regular matrix $A \in \mathbb{R}_{\max}$, but first, we introduce a new matrix form.

DEFINITION 4.4. Let $A \in \mathbb{R}_{\max}^{n \times n}$ be a regular matrix with corresponding graph $G(A)$, and let the collection $\{V_1(G(A)), \ldots, V_q(G(A))\}$ represent the partition formed on $V(G(A))$ by $\sim_C$. Recall that $\sim_C$ defines our strongly connected components. Define the matrix A_{rr}, for $r \in \{1, 2, \ldots, q\}$, as follows:

$$[A_{rr}]_{ij} = a_{ij}$$

for all $i, j \in V_r(G(A))$. Note that A_{rr} is either irreducible, or $A_{rr} = [\epsilon]$. Furthermore, let entries in the matrix A_{sr}, for $1 \leq s < r \leq q$, correspond to arcs from a node in $V_r(G(A))$ to a node in $V_s(G(A))$. Recall that $\mathbf{0}$ stands for the additive identity in the space $\mathbb{R}_{\max}^{m \times n}$; that is, $[\mathbf{0}]_{ij} = \epsilon$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$. Using these matrices, we can rewrite A in its *normal form* as

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & \cdots & A_{1q} \\ \mathbf{0} & A_{22} & \cdots & \cdots & A_{2q} \\ \mathbf{0} & \mathbf{0} & A_{33} & \cdots & A_{3q} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & A_{qq} \end{bmatrix}.$$

A corresponding partition can be made on entries in the vector $\mathbf{x}(k)$ from our recurrence relation. We will define new vectors $\mathbf{x}_i(k)$ for $i \in \{1, 2, \ldots, q\}$, not to be confused with $x_i(k)$, by which we denote the i^{th} element in the vector $\mathbf{x}(k)$.

DEFINITION 4.5. Let G be a digraph consisting of n vertices. Define a binary relation on $V(G)$ by $\sim_R$, where for $i, j \in \{1, 2, \ldots, n\}$, we have $j \sim_R i$ if and only if there exists a path of any length in G from node j to node i. Note that $\sim_R$ is not symmetric. Define the set of *direct predecessors* of i by

$$\pi(i) = \{j \in \{1, \ldots, n\} : (j, i) \in E(G)\}.$$

Let the set of *predecessors* of node i be

$$\pi^+(i) = \{j \in \{1, \ldots, n\} : j \sim_R i\},$$

and let $\pi^*(i) = \pi^+(i) \cup \{i\}$.

Example 4.1 provides an illustration of Definitions 4.4 and 4.5.

EXAMPLE 4.1. Consider the reducible graph $G(A)$ shown below.

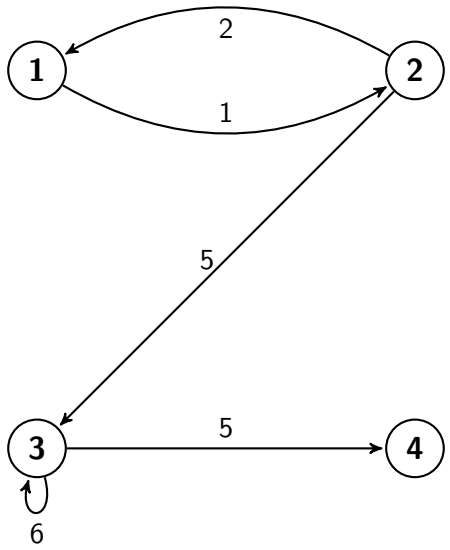


Figure 4.1

Its matrix

$$A = \begin{bmatrix} \epsilon & 2 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon & \epsilon \\ \epsilon & 5 & 6 & \epsilon \\ \epsilon & \epsilon & 5 & \epsilon \end{bmatrix}$$

is already in normal form, but for the sake of exposition, we will devise a new partition on $V(G(A))$. The strongly connected components of $G(A)$ will correspond to the following collection $\{V_1(G(A)), V_2(G(A)), V_3(G(A))\} = \{\{4\}, \{3\}, \{1, 2\}\}$. Hence,

$$A_{11} = [\epsilon], A_{12} = [5], A_{13} = \begin{bmatrix} \epsilon & \epsilon \end{bmatrix};$$
$$A_{22} = [6], A_{23} = \begin{bmatrix} \epsilon & 5 \end{bmatrix}; \text{and}$$
$$A_{33} = \begin{bmatrix} \epsilon & 2 \\ 1 & \epsilon \end{bmatrix}.$$

Hence, the alternate normal form of A is

$$\begin{bmatrix} \epsilon & 5 & \epsilon & \epsilon \\ \epsilon & 6 & \epsilon & 5 \\ \epsilon & \epsilon & \epsilon & 2 \\ \epsilon & \epsilon & 1 & \epsilon \end{bmatrix}.$$

Consider the recurrence relation $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$. Suppose we let $\mathbf{x}(0) = (3, 2, \epsilon, 4)^\intercal$. It follows that

$$\mathbf{x}_1(0) = [4],$$
$$\mathbf{x}_2(0) = [\epsilon], \text{and}$$
$$\mathbf{x}_3(0) = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Applying the mappings of Definition 4.5 to node 4 in $G(A)$, we have

$$\pi(4) = \{j \in \{1,2,3,4\} : (j,4) \in E(G(A))\} = \{3\},$$
$$\pi^+(4) = \{j \in \{1,2,3,4\} : j \sim_R 4\} = \{1,2,3\}, \text{and}$$
$$\pi^*(4) = \pi^+(4) \cup \{4\} = \{1,2,3,4\}.$$

We can now give a theorem which implies the existence of the cycle-time vector for regular tropical matrices.

THEOREM 4.4. *Let $A \in \mathbb{R}_{\max}^{n\times n}$ be a regular matrix. Consider the recurrence relations given by*

$$\mathbf{x}_i(k+1) = A_{ii} \otimes \mathbf{x}_i(k) \oplus \bigoplus_{j=i+1}^{q} A_{ij} \otimes \mathbf{x}_j(k) \qquad (4)$$

for $i \in \{1,2,\ldots,q\}$. Assume that A_{qq} is irreducible and that for $i \in \{1,2,\ldots,q-1\}$ either A_{ii} is irreducible or $A_{ii} = [\epsilon]$. Then there exist finite vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_q \in \mathbb{R}^n$ of suitable sizes and scalars $\xi_1, \xi_2, \ldots, \xi_q \in \mathbb{R}$ such that the sequences

$$\mathbf{x}_i(k) = \mathbf{v}_i \otimes \xi_i^{\otimes k} \qquad \text{for } i \in \{1,\ldots,q\}$$

satisfy (4) for all $k \geq 0$. The scalars $\xi_1, \xi_2, \ldots, \xi_q$ are determined by

$$\xi_i = \bigoplus_{\jmath \in \mathcal{H}_i} \xi_j \oplus \lambda_i$$

where $\mathcal{H}_i = \{j \in \{1,\ldots,q\} : j > i, A_{ij} \neq \mathbf{0}\}$.

We will not give a full proof of Theorem 4.4. To do so would involve the use of expressions known as inhomogenous recurrence relations, a topic beyond the scope of our work. Verifying the theorem requires a proof by induction that works backwards from q to 1. We will illustrate the base case $l = q$, as it is rather simple. Indeed, by the definition of

our recurrence relations in (4),

$$\begin{aligned}\mathbf{x}_q(k+1) &= A_{qq} \otimes \mathbf{x}_q(k) \oplus \bigoplus_{j=q}^{q} A_{qj} \otimes \mathbf{x}_j(k) \\ &= \mathbf{x}_q(k+1) \\ &= A_{qq} \otimes \mathbf{x}_q(k).\end{aligned}$$

Since we assumed A_{qq} to be irreducible, it has a unique eigenvalue λ_q. If we let $\xi_q = \lambda_q$ and $\mathbf{v}_q$ be an eigenvector associated with λ_q, then

$$\mathbf{x}_q(k) = \mathbf{v}_q \otimes \xi_q^{\otimes k}$$

for all $k \geq 0$. For a complete proof of Theorem 4.4, see Section 3.3.1 of [**2**].

The following is a corollary of Theorem 4.4 that demands the existence of the cycle-time vector for all regular max-plus matrices.

COROLLARY 4.5. *Consider the recurrence relation given by*

$$A \otimes \mathbf{x}(k) = \mathbf{x}(k+1), \tag{5}$$

where $A \in \mathbb{R}_{\max}^{n \times n}$ *is regular. There exist finite vectors* $\boldsymbol{\eta}, \mathbf{v} \in \mathbb{R}^n$ *such that*

$$A \otimes \mathbf{x}(k) = A \otimes (k \times \boldsymbol{\eta} + \mathbf{v}) = (k+1) \times \boldsymbol{\eta} + \mathbf{v}$$

for all $k \geq 0$.

PROOF. Using the renumbering suggested in Definition 4.4, we can express the recurrence relation given by (5) in terms of (4). We assume A to be regular, so that the matrices along the diagonal of the normal form of A satisfy Theorem 4.4. Consider the following rearrangement

of the sequences of Theorem 4.4:

$$\begin{aligned} \mathbf{x}_i(k) &= \mathbf{v}_i \otimes \xi_i^{\otimes k} && (6) \\ &= \mathbf{v}_i \otimes k \times \xi_i \\ &= k \times \mathbf{u}[\xi_i] + \mathbf{v}_i, && (7) \end{aligned}$$

where $\mathbf{u}[c]$ denotes the scalar multiple of the tropical unit vector given by

$$\begin{aligned} \mathbf{u}[c] &= c \otimes \mathbf{u} \\ &= c \otimes (0, 0, \ldots, 0)^\intercal. \end{aligned}$$

Since, by Theorem 4.4, (6) holds for all $k \geq 0$, so too does (7). Hence, the vectors $\boldsymbol{\eta}' = (\mathbf{u}[\xi_1], \mathbf{u}[\xi_2], \ldots, \mathbf{u}[\xi_q)^\intercal$ and $\mathbf{v}' = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_q)^\intercal$ satisfy the above corollary for A in its normal form. An appropriate reordering (working backwards from the partitioning of $V(G(A))$ under $\sim_C$) of $\boldsymbol{\eta}'$ and $\mathbf{v}'$ would yield vectors $\boldsymbol{\eta}, \mathbf{v} \in \mathbb{R}^n$ satisfying the corollary for A. □

By Theorem 4.2 and Corollary 4.5, for any regular tropical matrix $A \in \mathbb{R}_{\max}^{n \times n}$, there exists a generalized eigenmode $(\boldsymbol{\eta}, \mathbf{v}) \in \mathbb{R}^n \times \mathbb{R}^n$ with unique cycle-time vector $\boldsymbol{\eta}$. It follows from our proof of Corollary 4.5, that the cycle-time vector of a reducible matrix A would depend upon the eigenvalues of the strongly connected components of $G(A)$, as (7) holds for all $k \geq 0$. However, as our definition of the scalar ξ_i in Theorem 4.4 would indicate, expressing the eigenmode of a regular matrix A is not as simple as finding the eigenvalues and eigenspaces of each A_{ii} matrix and rearranging them in $\boldsymbol{\eta}$ and $\mathbf{v}$ accordingly. Each entry η_i depends upon the interactions between the strongly connected

component to which i belongs and $G(A)$'s other maximal strongly connected subgraphs. The mapping $\pi^*(i)$ gives us a systematic method for tracking these interactions. The convergence of ξ_i, λ_i, and $\pi^*(i)$ in determining each η_i for the cycle time vector of A is revealed by the next theorem.

THEOREM 4.6. *Consider the recurrence relation* $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$ *(*$k \geq 0$*) defined by the regular matrix* $A \in \mathbb{R}_{\max}^{n \times n}$ *and arbitrary initial condition* $\mathbf{x}(0) \in \mathbb{R}^n$. *Let* $V_1(G(A)), V_2(G(A)), \ldots, V_q(G(A))$ *be the equivalence classes on* $V(G(A))$ *under* $\sim_C$, *and let*

$$\boldsymbol{\eta} = \lim_{k \to \infty} \frac{A^{\otimes k} \otimes \mathbf{x}(0)}{k}$$

be the cycle-time vector of A. *If* $[j]$ *denotes the equivalence class to which node* $j \in V(G(A))$ *belongs, then for all* $j \in V(G(A))$

$$\xi_{[j]} = \bigoplus_{i \in \pi^*(j)} \lambda_{[i]}, \text{ and}$$

$$\lim_{k \to \infty} \frac{A^{\otimes k} \otimes \mathbf{x}(0)}{k} = \bigoplus_{i \in \pi^*(j)} \lambda_{[i]}.$$

Note that our use of the $[j]$ notation in Theorem 4.6 implies that if node i and node j belong to the same strongly connected component of $G(A)$, then $\eta_i = \eta_j$, where $\boldsymbol{\eta}$ is the cycle-time vector of A. The following example implements Theorem 4.6 to calculate the cycle-time vector of a reducible matrix.

EXAMPLE 4.2. Let $G(A)$ be the reducible graph pictured in Figure 3.8. We will use the same indexing of the strongly connected components as in Example 4.1. Our graph is simple enough so that each $\lambda_{[i]}$

is easily calculated:

$$\lambda_{[1]} = \lambda_{[2]} = \frac{3}{2};$$
$$\lambda_{[3]} = 6;$$
$$\lambda_{[4]} = \epsilon.$$

By Corollary 4.5,

$$\xi_{[1]} = \lambda_{[1]} \oplus \lambda_{[2]} = \frac{3}{2} \oplus \frac{3}{2} = \frac{3}{2}$$
$$\xi_{[2]} = \lambda_{[1]} \oplus \lambda_{[2]} = \frac{3}{2} \oplus \frac{3}{2} = \frac{3}{2}$$
$$\xi_{[3]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} = \frac{3}{2} \oplus \frac{3}{2} \oplus 6 = 6$$
$$\xi_{[4]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{3}{2} \oplus \frac{3}{2} \oplus 6 \oplus \epsilon = 6.$$

We conclude that the vector $\boldsymbol{\eta} = (\frac{3}{2}, \frac{3}{2}, 6, 6)^\intercal$ is the cycle-time vector of A.

Hence, we have a method for finding the cycle-time vector of a regular matrix. In the next section, we will present an algorithm that incorporates the ideas of Theorem 4.6 into a method for computing the generalized eigenmode of a regular matrix.

2. Howard's Algorithm

Published in 1960, a decade before the development of the max-plus and min-plus algebras, Ronald A. Howard's Policy Improvement Algorithm has since been applied outside its original context of solving Markov decision problems to the determination of generalized eigenmode for tropical matrices. Before detailing Howard's method, we define the term policy.

DEFINITION 4.6. Let $A \in \mathbb{R}_{\max}^{n\times n}$ be a regular matrix and $G(A)$ be its corresponding graph. A *policy* Π is a mapping $\Pi : V(G(A)) \to E(G(A))$ that assigns an arc $\Pi_i \in E(G(A))$ to each node $i \in V(G(A))$, where node i is the end of arc Π_i. The *policy matrix*, denoted A^{Π}, contains the arc weights corresponding to $\Pi(1), \Pi(2), \ldots, \Pi(n)$ with all other entries equal to ϵ.

Note that policies are constructed so that there is exactly one finite entry in each row of A^{Π}. Therefore, each node in $G(A^{\Pi})$ has exactly one direct predecessor; we denote the direct predecessor of j in $G(A^{\Pi})$ with $\pi_{\Pi}(j)$. Howard's Algorithm requires its inputs to be regular because were A to contain a row with only infinite entries, we would not be able to construct a policy for A.

HOWARD'S ALGORITHM. *Let $A \in \mathbb{R}_{\max}^{n\times n}$ be a regular matrix with the corresponding graph $G(A)$, and let $\mathbf{v} = (0, 0, \ldots, 0)^{\intercal}$. We compute the generalized eigenmode of A with the following algorithm:*

(1) From $G(A)$, construct a Π.

(2) Find a circuit γ in $G(A^{\Pi})$.

(3) Compute the average weight of γ and denote it by $\bar{\eta}_{\gamma}$.

(4) Choose a node in γ, say node j, and fix $\eta_j := \bar{\eta}_{\gamma}$; let v_j keep the value it had previously.

(5) Visit all nodes in $G(A^{\Pi})$ that are reachable from j. If node i is visited in the process, set $\eta_i := \bar{\eta}_{\gamma}$, and compute v_i from the expression

$$v_i = a_{i\pi_{\Pi}(i)} - \bar{\eta}_{\gamma} + v_{\pi_{\Pi}(i)}.$$

(6) If there remain nodes that cannot be reached by j through arcs in $G(A^{\Pi})$, repeat steps (2) through (5) with these nodes and

their associated arcs in $G(A^{\Pi})$ *until values of* $\boldsymbol{\eta}$ *and* $\mathbf{v}$ *are determined.*

(7) Determine the set $I_1 = \left\{ i \in V(G(A)) : \eta_i < \max_{(j,i)\in E(G(A))} \eta_j \right\}$.

- *If* $I_1 = \emptyset$, *compute* $E(\bar{A}) = \{(j,i) \in E(G(A)) : \eta_j = \eta_i\}$, *and continue with step (8).*
- *If* $I_1 \neq \emptyset$, *determine for all* $i \in I_1$ *the sets*

$$E(G(A))_i^1 = \left\{ (k,i) \in E(G(A)) : \eta_k = \max_{(j,i)\in E(G(A))} \eta_j \right\},$$

and return to step (2), defining your new policy as

$$\Pi_i' := \begin{cases} (k,i) & \text{for some } (k,i) \in E(G(A))_i^1 \text{if } i \in I_1, \\ \Pi_i & \text{if } i \notin I_1. \end{cases}$$

(8) Determine the set $I_2 = \left\{ i \in V(G(A)) : v_i < \max_{(j,i)\in E(G(\bar{A}))} (a_{ij} + v_j - \eta_j) \right\}$.

- *If* $I_2 = \emptyset$, *then the algorithm is complete, and our current values of* $\boldsymbol{\eta}$ *and* $\mathbf{v}$ *constitute the generalized eigenmode of* A.
- *If* $I_2 \neq \emptyset$, *then determine for all* $i \in I_2$ *the sets*

$$E(G(A))_i^2 = \left\{ (k,i) \in E(G(A)) : a_{ik} + v_k - \eta_k = \max_{(j,i)\in E(G(\bar{A}))} (a_{ij} + v_j - \eta_j) \right\},$$

and return to step (2), defining your new policy as

$$\Pi_i' := \begin{cases} (k,i) & \text{for some } (k,i) \in E(G(A))_i^2 \text{if } i \in I_2, \\ \Pi_i & \text{if } i \notin I_2. \end{cases}$$

The method of policy improvement is much more complex than anything we have encountered up until this point, but the following simple example should be illuminating.

EXAMPLE 4.3. The regular matrix $A \in \mathbb{R}_{\max}^{3\times 3}$ is given by

$$A = \begin{bmatrix} 3 & \epsilon & \epsilon \\ 1 & \epsilon & 6 \\ \epsilon & 2 & \epsilon \end{bmatrix}.$$

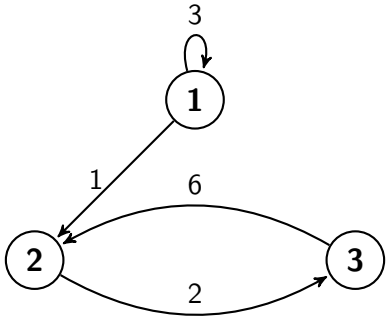


Figure 4.2

We can see from the graph in Figure 4.1 that $G(A)$ is reducible; node 1 cannot be reached from either node 2 or node 3. We define our first policy $\Pi(1) : V(G(A)) \to E(G(A))$ by $\Pi_1(1) = (1,1)$, $\Pi_2(1) = (3,2)$, and $\Pi_3(1) = (2,3)$. Thus,

$$A^{\Pi(1)} = \begin{bmatrix} 3 & \epsilon & \epsilon \\ \epsilon & \epsilon & 6 \\ \epsilon & 2 & \epsilon \end{bmatrix}.$$

Next, we fix $\gamma := (1,1)$, so that $\bar{\eta}_\gamma := 3$, and, thus, $\eta_1^{\Pi(1)} = 3$. Furthermore, since this is our first iteration of the algorithm, $v_1^{\Pi(1)} = 0$. Since we eliminated $a_{21} = 1$ from A in defining our policy matrix, nodes 2 and 3 cannot be reached, so we reset $\gamma := (2,3) \circ (3,2)$. Hence, $\bar{\eta}_\gamma := \eta_2^{\Pi(1)} = 4$, and $v_2^{\Pi(1)} = 0$. Node 3 is reached within our new γ,

allowing us to set $\eta_3^{\Pi(1)} = 4$ and to compute

$$\begin{aligned} v_3^{\Pi(1)} &= a_{32} - \bar{\eta}_\gamma + v_2 \\ &= 2 - 4 + 0 \\ &= -2. \end{aligned}$$

We proceed to step (7) with $\boldsymbol{\eta}^{\Pi(1)} = (3, 4, 4)^\intercal$ and $\mathbf{v}^{\Pi(1)} = (0, 0, -2)^\intercal$. Here, we are looking for a node i in $V(G(A)$ such that the corresponding value $\eta_i^{\Pi(1)}$ is less than some entry in $\boldsymbol{\eta}^{\Pi(1)}$ that corresponds to a direct predecessor of i. We have $\eta_1^{\Pi(1)} < \eta_2^{\Pi(1)}, \eta_3^{\Pi(1)}$, but neither node 2 nor node 3 is a direct predecessor of node 1. Hence, $I_1 = \emptyset$, and because $\eta_2^{\Pi(1)} = \eta_3^{\Pi(1)}$,

$$E(G(\bar{A})) = \{(2, 3), (3, 2)\}.$$

The set I_2 cannot contain node 1, as it is not traversed by an arc in $E(G(\bar{A}))$. Consider the following calculations:

$$\begin{aligned} a_{23} + v_2^{\Pi(1)} - \eta_3^{\Pi(1)} &= 6 + (-2) - 4 = 0 \\ a_{32} + v_3^{\Pi(1)} - \eta_2^{\Pi(1)} &= 2 + 0 - 4 = -2. \end{aligned}$$

From the above expressions, it is easily seen that neither node 2 nor node 3 belong in I_2. Therefore, the algorithm is complete, and we have found the generalized eigenmode of A to be

$$(\boldsymbol{\eta}, \mathbf{v}) = ((3, 4, 4)^\intercal, (0, 0, -2)^\intercal).$$

We verify this with the following computation:

$$\begin{bmatrix} 3 & \epsilon & \epsilon \\ 1 & \epsilon & 6 \\ \epsilon & 2 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 3 \\ 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix},$$

so in fact, $A \otimes (0 \times \boldsymbol{\eta} + \mathbf{v}) = 1 \times \boldsymbol{\eta} + \mathbf{v}$.

3. Extending Karp's Algorithm

We continue to explore the computation of the generalized eigenmode of reducible matrices through the analytical lens established in [**2**]. Hence, our work will be in the max-plus algebra. However, the purpose of this section is to extend the work of Heidergott, Olsder, and van der Woude rather than provide mere exposition. More specifically, we hope to shed some light on how and why Karp's Algorithm is inconsistent when finding the maximal average circuit weight of reducible systems. Ultimately, we will show that Karp's Algorithm does, in fact, reveal information about reducible matrices and can be extended to compute the generalized eigenmode, $\boldsymbol{\eta}$.

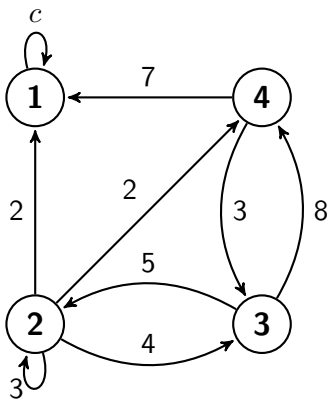
Is there any connection between our choice of j in performing Karp's Algorithm on a regular, reducible matrix A and the corresponding entry in the cycle-time vector of A? Put another way, when does $\lambda = \eta_j$, where λ is our algorithm output? If so, what are the conditions on $G(A)$ and A that must be satisfied? The following analysis elaborates upon Examples 5.1.2 and 5.1.3 from *Max Plus at Work*, and in doing so, attempts to answer these questions.

The following three examples contain distinct graphs with important similarities. First, note that the strongly connected components of all three networks are identical. For reference, we may let $V_1(G) = \{1\}$ and $V_2(G) = \{2, 3, 4\}$ represent the partition of vertices under $\sim_C$ for each graph. Second, consider the arc weight of $(1, 1)$, given as $c \in \mathbb{R}$, in each example to be a tool of experimentation. We will demonstrate how fluctuations in the value of c alter the cycle-time vector and the output of Karp's Algorithm. In the first two examples, we consider the cases when $|(1, 1)|_W = 1$, and $|(1, 1)|_W = 6$. However, it is important to note that the overarching consequences of the former case hold for all $c < \frac{11}{2}$, and those of the latter for all $c > \frac{11}{2}$.

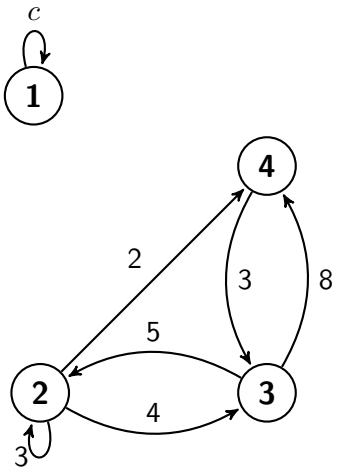
EXAMPLE 4.4. Let $A \in \mathbb{R}^{4\times4}_{\max}$ be defined as

$$A = \begin{bmatrix} c & 2 & \epsilon & 7 \\ \epsilon & 3 & 5 & \epsilon \\ \epsilon & 4 & \epsilon & 3 \\ \epsilon & 2 & 8 & \epsilon \end{bmatrix},$$

We represent $G(A)$ and its strongly connected components in Figure 4.1 below.



$G(A)$



$G_1(A)$

$G_2(A)$

Figure 4.3

First, consider the case where $c = 1$. We will calculate the cycle-time vector of A. Note that the equivalence classes of $V(G(A))$ under $\sim_C$ are $V_1(G(A)) = \{1\}$ and $V_2(G(A)) = \{2, 3, 4\}$. Obviously, $\lambda_{[1]} = 1$, as $G_1(A)$ is a one node graph with a loop of weight 1. Considering the three elementary circuits in $G_2(A)$, we find that the maximal average circuit weight is $\lambda_{[2]} = \lambda_{[3]} = \lambda_{[4]} = \frac{11}{2}$. By Theorem 4.6, we have

$$\xi_{[1]} = \bigoplus_{i \in \pi^*(1)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 1 \oplus \frac{11}{2} = \frac{11}{2}$$

$$\xi_{[2]} = \bigoplus_{i \in \pi^*(2)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2}$$

$$\xi_{[3]} = \bigoplus_{i \in \pi^*(3)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2}$$

$$\xi_{[4]} = \bigoplus_{i \in \pi^*(4)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2},$$

and, thus, $\boldsymbol{\eta} = (\frac{11}{2}, \frac{11}{2}, \frac{11}{2}, \frac{11}{2})^\intercal$, which means that our system has an eigenvalue $\lambda = \frac{11}{2}$. When performing Karp's Algorithm, whether or not

we find the correct value for λ would depend on our choice of j. Note that the graph's critical circuit can be expressed as $\gamma = (3,4) \circ (4,3)$, and the success of Karp's Algorithm is contingent upon node j's ability to reach a node in the critcal circuit. Node 1 does not have access, so to speak, to this circuit, as there is no path from node 1 to node 3 or node 4. Thus, if we choose $j = 1$ and run Karp's Algorithm, we find that $\mathbf{x}(k) = (k, \epsilon, \epsilon, \epsilon)^\intercal$ for $0 \leq k \leq 4$. Note that as a convention, we have $\epsilon - \epsilon = 0$ (in both the max-plus and min-plus structures). Hence, step (3) of the algorithm yields

$$\min\left\{\frac{4-0}{4-0}, \frac{4-1}{4-1}, \frac{4-2}{4-2}, \frac{4-3}{4-3}\right\} = \min\{1,1,1,1\} = 1; \quad i = 1$$

$$\min\left\{\frac{\epsilon-\epsilon}{4-0}, \frac{\epsilon-\epsilon}{4-1}, \frac{\epsilon-\epsilon}{4-2}, \frac{\epsilon-\epsilon}{4-3}\right\} = \min\{0,0,0,0\} = 0, \quad i = 2,3,4$$

and $\lambda = \max\{0,1\} = 1$, a result equal to neither the graph's maximal mean circuit weight, nor an eigenvalue of A. Since nodes 2, 3, and 4 exist within the strongly connected component containing the critical circuit, running Karp with $j = 2, 3, \text{or } 4$ would yield a correct result: $\lambda = \frac{11}{2}$.

Consider the case when $c = 6$. Now, we have a circuit $(1,1)$ such that $|(1,1)|_W = 6 > \frac{11}{2}$. With a change in the maximal average circuit weight of $G(A)$, so too changes the effectiveness of Karp's Algorithm. Before running Karp, we calculate the cycle-time vector of A:

$$\xi_{[1]} = \bigoplus_{i \in \pi^*(1)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 6 \oplus \frac{11}{2} = 6;$$

$$\xi_{[2]} = \bigoplus_{i \in \pi^*(2)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2};$$

$$\xi_{[3]} = \bigoplus_{i \in \pi^*(3)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2};$$

$$\xi_{[4]} = \bigoplus_{i \in \pi^*(4)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2}.$$

Hence, the cycle-time vector of our new matrix is $\boldsymbol{\eta} = (6, \frac{11}{2}, \frac{11}{2}, \frac{11}{2})^\intercal$. When we let $c = 1$, the weight of the circuit $(1, 1)$ was absent from our cycle-time vector because nodes 2, 3, and 4, all belonging to a strongly connected component with maximal circuit mean greater than $|(1,1)|_W = 1$, preceded node 1. For the case of $c = 6$, we still have $\xi_1 = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]}$, but this time $\lambda_{[1]} = 6$ attains the maximum. Thus, we see from the contrast of results from when $c = 1$ to $c = 6$ that the values of our cycle-time vector depend upon communication between the strongly connected components of our graph. This will be something to consider when extending Karp's Algorithm to compute cycle-time vectors.

Karp's Algorithm now "works" for all j because node 1, the vertex over which the critical circuit traverses, can be reached from all other nodes in $G(A)$. We show the steps of Karp's Algorithm for $j = 1$, the case for which algorithm failed to correctly calculate the maximal average circuit weight of $G(A)$ when $a_{11} = c = 1$. This time we may express our catalogue of maximum path weights as $\mathbf{x}(k) = (6k, \epsilon, \epsilon, \epsilon)$ for $0 \leq k \leq 4$. Hence, step (3) appears as

$$\min\left\{\frac{24-0}{4-0}, \frac{24-6}{4-1}, \frac{24-12}{4-2}, \frac{24-18}{4-3}\right\} = \min\{6,6,6,6\} = 6 \quad i = 1$$

$$\min\left\{\frac{\epsilon-\epsilon}{4-0}, \frac{\epsilon-\epsilon}{4-1}, \frac{\epsilon-\epsilon}{4-2}, \frac{\epsilon-\epsilon}{4-3}\right\} = \min\{0,0,0,0\} = 0, \qquad i = 2,3,4$$

so that $\lambda = \max\{6, 0\} = 6$. However, when we say Karp's Algorithm "works," we mean that it successfully computes the maximal average circuit weight of $G(A)$. From a linear algebraic standpoint, this value tells us very little about the behavior of the recurrence relation $A \otimes \mathbf{x}(k) = \mathbf{x}(k+1)$ as k approaches infinity. In this sense, Karp's Algorithm fails, as it cannot reveal to us the fact that $\boldsymbol{\eta} = (6, \frac{11}{2}, \frac{11}{2}, \frac{11}{2})^{\intercal}$.

EXAMPLE 4.5. Consider the max-plus matrix whose graph is depicted in Figure 4.3:

$$A = \begin{bmatrix} c & \epsilon & \epsilon & \epsilon \\ 3 & 3 & 5 & \epsilon \\ \epsilon & 4 & \epsilon & 3 \\ \epsilon & 2 & 8 & \epsilon \end{bmatrix}.$$

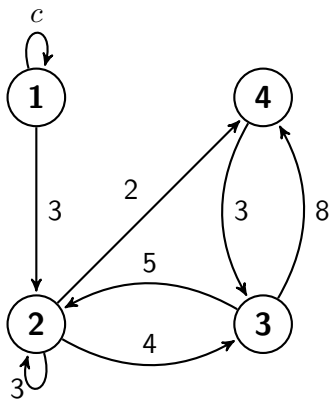


Figure 4.4

Note that the strongly connected components of $G(A)$ are the same as those of the graph in Example 4.4. This graph differs from that of Figure 4.2 because our strongly connected component $V_1(G(A))$ now precedes nodes in $V_2(G(A))$, rather than vice versa. We consider how this adjustment alters the cycle-time vector of A.

Let $c = 1$. It follows that $\lambda_{[1]} = 1$, and $\lambda_{[2]} = \lambda_{[3]} = \lambda_{[4]} = \frac{11}{2}$. Then

by Theorem 4.6,

$$\xi_{[1]} = \bigoplus_{i\in\pi^*(1)} \lambda_{[i]} = \lambda_{[1]} = 1,$$

$$\xi_{[2]} = \bigoplus_{i\in\pi^*(2)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 1 \oplus \frac{11}{2} = \frac{11}{2},$$

$$\xi_{[3]} = \bigoplus_{i\in\pi^*(3)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 1 \oplus \frac{11}{2} = \frac{11}{2}, \text{and}$$

$$\xi_{[4]} = \bigoplus_{i\in\pi^*(4)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 1 \oplus \frac{11}{2} = \frac{11}{2}.$$

Thus, $\boldsymbol{\eta} = (1, \frac{11}{2}, \frac{11}{2}, \frac{11}{2})^\intercal$. Although $\lambda_{[i]} > \lambda_{[1]}$ for $i = 2, 3, 4$, the second equivalence class of vertices under $\sim_C$, seen as a strongly connected component of $G(A)$, has no influence over node one. In other words, nodes 2, 3, and 4 are not predecessors of node 1, so the value of the critical circuit in $G_2(A)$ does not affect the value of η_1. However, now all nodes are predecessors of the critical circuit $(3, 4) \circ (4, 3)$, so Karp's Algorithm returns $\lambda = \frac{11}{2}$ for all possible choices of j. To answer our motivating question, Karp's Algorithm has, once again, failed to satisfy $\lambda = \eta_j$ for $j = 1, 2, 3, 4$.

We shall now consider the case when $c = 6$. Consequently, $\lambda_{[1]} = 6$, and $\lambda_{[2]} = \lambda_{[3]} = \lambda_{[4]} = \frac{11}{2}$, and

$$\xi_{[1]} = \bigoplus_{i\in\pi^*(1)} \lambda_{[i]} = \lambda_{[1]} = 6,$$

$$\xi_{[2]} = \bigoplus_{i\in\pi^*(2)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 6 \oplus \frac{11}{2} = 6,$$

$$\xi_{[3]} = \bigoplus_{i\in\pi^*(3)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 6 \oplus \frac{11}{2} = 6, \text{and}$$

$$\xi_{[4]} = \bigoplus_{i\in\pi^*(4)} \lambda_{[i]} = \lambda_{[1]} \oplus \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = 6 \oplus \frac{11}{2} = 6.$$

The matrix now has a max-plus eigenvalue of $\lambda = 6$. However, since node 1 cannot be reached by any vertex other than itself, $j = 1$ is the only case where Karp's Algorithm correctly computes the value of η_j. For $j = 2, 3,$ and 4, the output is $\frac{11}{2}$, as the circuit $(3,4) \circ (4,3)$ is the critical circuit in $G_2(A)$, and there is no path from this component into $G_1(A)$. Once again, Karp's Algorithm has failed to fully reveal the cycle-time vector due to the nature of communication between strongly connected components.

EXAMPLE 4.6. In our final example, we let the condensed graph of $G(A)$ be empty. That is, $G(A)$ consists of only strongly connected components with no communication between them. The graph of

$$A = \begin{bmatrix} c & \epsilon & \epsilon & \epsilon \\ \epsilon & 3 & 5 & \epsilon \\ \epsilon & 4 & \epsilon & 3 \\ \epsilon & 2 & 8 & \epsilon \end{bmatrix}$$

is pictured in the figure below.

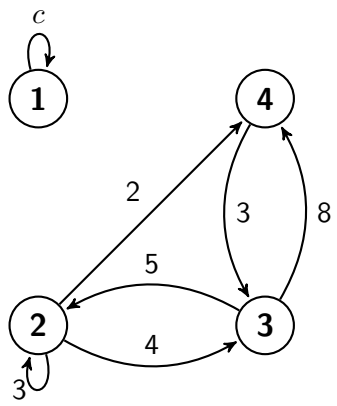


Figure 4.4

Note that, for this example, c will be treated as an arbitrary, finite value. Silencing communication between the strongly connected components of our graph has ensured that for each node in $G(A)$, the corresponding entry in the cycle-time vector of A will be the average weight of the critical circuit in its respective strongly connected component. Indeed,

$$\xi_{[1]} = \bigoplus_{i\in\pi^*(1)} \lambda_{[i]} = \lambda_{[1]} = c,$$

$$\xi_{[2]} = \bigoplus_{i\in\pi^*(2)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2},$$

$$\xi_{[3]} = \bigoplus_{i\in\pi^*(3)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2}, \text{and}$$

$$\xi_{[4]} = \bigoplus_{i\in\pi^*(4)} \lambda_{[i]} = \lambda_{[2]} \oplus \lambda_{[3]} \oplus \lambda_{[4]} = \frac{11}{2}.$$

For all $c \in \mathbb{R}$, the cycle-time vector of A attains the value $(c, \frac{11}{2}, \frac{11}{2}, \frac{11}{2})^\intercal$. Moreover, if we run Karp's Algorithm with $j = 1$, we find that, as was the case in Example 4.4, node 1 cannot reach any other, so the only circuit the algorithm can track is the loop of length 1 and weight c. Hence, $\lambda = c = \eta_1$. Fixing j to 2, 3, or 4 and executing the algorithm will yield $\lambda = \frac{11}{2} = \eta_j$. This is because, with node j as our basis of operations, Karp's Algorithm will never even consider the circuit $(1, 1)$ as a candidate for the critical circuit. Thus, the only circuits that remain relevant are those traversing nodes of $V_2(G(A))$, and the algorithm will find the maximum average circuit weight of this subgraph to be $\frac{11}{2}$. We, therefore, conclude that it is the absence of a condensed graph of $G(A)$ that allows Karp's Algorithm to find the correct values of η_j for $j = 1, 2, 3, 4$.

That Karp's Algorithm requires a graph to have no condensed affiliate in order to properly compute the individual entries of $\boldsymbol{\eta}$ indicates that, in it's original form, it is an impractical method for computing the cycle-time vector of a regular max-plus matrix. However, we can use this notion of limited workability to extend the algorithm.

An Extension of Karp's Algorithm. *Consider the regular matrix $A \in \mathbb{R}_{\max}^{n\times n}$ and its corresponding graph $G(A) = \{V(G(A)), E(G(A)), I_G, K_G\}$. Let $\boldsymbol{\eta}$ represent the cycle-time vector of A. The following steps compute the vector $\boldsymbol{\eta}$.*

(1) Establish a partition on $V(G(A))$ under $\sim_C$, so that its equivalence classes are $V_1(G(A), V_2(G(A)), \ldots, V_q(G(A))$. If for all $\ell, p \in V(G(A))$, $\ell \sim_c p$, proceed with Karp's Algorithm to produce $\eta_j = \lambda$ for all $j \in \{1, 2, \ldots, n\}$. Otherwise, proceed to step (2).

(2) Using Karp's Algorithm, compute $\lambda_{[i]}$, the maximal average circuit weight of $G_i(A)$, for $i \in \{1, 2, \ldots, q\}$.

(3) Compute $\eta_j = \bigoplus_{i\in\pi^*(j)} \lambda_{[i]}$ for $j \in \{1, 2, \ldots, n\}$, so that $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_n)^\intercal$.

The construction of this proposed algorithm, using a formal language such as Python, remains an open problem. To create such an algorithm, we would employ Tarjan's depth-first search technique for identifying strongly connected components. Depth-first search techniques could also be employed to find the predecessors of a given node outside its equivalence class under $\sim_C$ (this is, in essence, the problem of constructing a condensed graph) [**9**].

4. Conclusions

Throughout our work, we hope to have shown that tropical linear algebra and graph theory are utterly insoluble. We first saw this fact in Chapter 1, where analyzing the paths of a weighted digraph motivated the computation of min-plus matrix powers. It became even more apparent in Chapter 2, where without the graph-theoretic infrastructure embedded in tropical matrices, we could not have proven the existence and uniqueness of eigenvalues for irreducible matrices (The Tropical Eigenvalue Theorem).

With the inextricability of graph theory and linear algebra being our analytical impetus, Karp's Algorithm became a natural object of study. The computations for carrying out Karp on a tropical matrix (or a digraph depending upon your perspective) are linear algebraic in nature. However, as we saw in Chapter 3, to understand fully what it is these computations are working towards, seeking out a node in the graph that is contained in a critical circuit, we require a graph theoretic approach.

Indeed, through our graph theoretic analysis of Karp's Algorithm we were able to find its shortcomings when computing entries of the cycle-time vector for regular, reducible max-plus matrices. As was revealed in Theorem 4.6, the values of entries in the vector $\boldsymbol{\eta}$ hinge upon communication between the strongly connected components of the matrix's corresponding digraph. Howard's method of policy correction, with its construction of the sets I_1 and I_2, cleverly takes this fact into account when calculating the generalized eigenmode. With regard to Karp, if the basis of operations established by our choice of j does not reach a graph's critical circuit, or can only reach a critical circuit whose constituent nodes are not predecessors of another strongly

connected component, the output of the algorithm is relatively useless. However, we saw that in the special case of a reducible matrix whose corresponding condensed graph was empty that, given a starting position j, Karp's Algorithm yielded $\lambda = \eta_j$, the j^{th} entry in the cycle-time vector. We can take advantage of this fact to construct a method that uses Karp's Algorithm to compute the maximal average circuit weight of the strongly connected components of a graph. Then, having established these components with a depth-first search, we track the interactions between components in the condensed graph, and use this information to compute the various entries of $\boldsymbol{\eta}$. The translation of this algorithm into a formal language remains an open problem. Another matter worth investigating is the viability of describing periodicity of min-plus matrices with the cycle-time vector.

Appendix A: Sage Min-plus and Max-plus Package Descriptions

The following packages were constructed for experimental purposes. In order to carry out our analyses of Karp's Algorithm and the Power Algorithm, these methods had to be run and rerun, so as to see how adjustments to the properties of input matrices would alter outputs. Some of these functions contain commands that, despite making overall performance less efficient, we found experimentally beneficial. Details about such instances can be found in the descriptions of individual functions below.

We chose to develop our code in Sage, an open-source platform, due to the current absence of tropical linear algebra tools. Currently, Sage does include max-plus and min-plus semirings [**10**], but its source code does not permit elements of a semiring (rings only) to be matrix entries. We had originally hoped to alter the source code so that linear algebra could be performed in the spaces $\mathbb{R}_{\max}^{n\times n}$ and $\mathbb{R}_{\min}^{n\times n}$, but the development of this project's more theoretical aspects posed too many temporal constraints. Constructing tropical linear algebra in Sage is a project we plan to carry out in the coming semester.

To circumvent Sage's lack of tropical infrastructure, we chose to treat Python lists within lists as matrix objects. For example, the

matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

would appear in our functions as [[a, b], [c, d]]. Vectors are simply lists, so we would represent the vector $\mathbf{x} = (x_1, x_2)^\intercal$ by [x_1, x_2]. The Python string 'id' serves as our additive identity in both packages. This lack of infrastructure lead to some inefficiencies, and since these functions are cumulative (for example, our function for computing tropical dot products is incorporated into our matrix multiplication function), inefficiencies propagate throughout the package. Certainly, the development of a Sage matrix package that permits semiring entries would remedy such inefficiencies. Another improvement that could be made is the incorporation of the min-plus and max-plus packages into one tropical package, perhaps through optional function parameters. What follows is a catalogue of our functions, including their descriptions , specifications of the parameters inputted, and examples of their execution.

Min-Plus Package

`minplus_addition(a, *b)`

Description: Returns the min-plus sum of any finite number of elements in $\mathbb{R}_{\min}$. This function is part of the foundation of our package. The parameter *b is stored as a tuple which we then convert to a list so that we have a mutable Python object. The function goes through a series of conditionals to determine which parameters, if any, are our additive identity. If a == 'id', then it is ignored. If *b contains the additive identity, then the list.pop() method is used to eliminate all such occurrences. Eventually, we consider only

finite entries, over which we use Python's built-in min() function. Of course, if all entries are infinite, then the function returns 'id'.

Parameters:

a - A semiring element

*b - Any number of semiring elements

Example:

```
minplus_addition(17, 'id')

17

minplus_addition(17, 'id', 4, 'id')

4
```

```
tropical_multiplication(a, *b)
```

Description: The multiplication function forms the other half of our package's foundation. As with addition, this function receives any finite number of $\mathbb{R}_{\min}$ elements. Conditionals search for the additive identity, and if it is at all present, our output is 'id'. Otherwise, Python's built-in sum() function "multiplies" finite entries.

Parameters:

a - A semiring element

*b - Any number of semiring elements

Example:

```
tropical_multiplication(7,'id',2)

'id'

tropical_multiplication(7,2)
```

```
9
```

`minplus_list_addition(A, B)`

Description: Returns the min-plus sum of two matrices. The function begins with two conditionals that make sure our matrix inputs are of equal dimension. Using Python's len() function, we test the length of the outer list to make sure each matrix has the same number of rows. Then, we verify that the length of the first list entries within the outer lists match in length. In this regard, the function is limited in that it only makes certain that the dimensions of the first columns of each matrix match. Using nested for loops, the function iterates over each matrix row, constructing an empty list for each row of input A. The next for loop iterates over each element within the matrix rows and appends the min-plus sum of corresponding elements to the aforementioned empty list. These altered lists then become the rows of the output matrix.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

B - A matrix with semiring elements inputted as a list of lists

Example:

```
minplus_list_addition([[23, 'id', 4], [-4, 5, -6], [2, 3, -1]],
                      [[6, 7, 3], [1, 10, 'id'], [1, 8, 6]])


[[6, 7, 3], [-4, 5, -6], [1, 3, -1]]
```

`minplus_dot_product(x, y)`

Description: This function returns the min-plus dot product of two vectors. First, a conditional checks to makes sure the vectors are of equal dimension. Next, a for loop iterates over a list of integers equal in length to the input

vectors to perform min-plus multiplication on corresponding elements. After each min-plus product is appended to a list called "dot_product", a for loop takes the min-plus sum of the "dot_product" list.

Parameters:

x - A vector with semiring elements

y - A vector with semiring elements

Example:

```
minplus_dot_product([13, 8, 4], [14, 16, 'id'])


24
```

`list_transpose(A)`

Description: This function plays a crucial role in our matrix multiplication functions. A for loop establishes empty lists that will be the new rows of $A^{\intercal}$. A second for loop appends the element A[i][j] to the position A[j][i] in our new lists of lists.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

Example:

```
list_transpose([[1, 2, 3], ['id', 3, 5]])


[[1, 'id'], [2, 3], [3, 5]]
```

`minplus_list_multiplication(A, B)`

Description: Returns the min-plus product of two matrices. A conditional is given to make sure that the number of columns in the first entry equals the number of rows in the second entry. A second conditional performs multiplication for the special case where B is a matrix with one column. Refining our package would include altering this function so that it could accept vector

entries, rather than treating one-column matrices and vectors as different objects. In both cases, where len(B) == 1 and len(B)>1, the function proceeds to take the min-plus dot product of the rows of A with the column(s) of B and appends these products to lists that are then appended to yet another list. Note that for the third conditional, when B has more than one column, the transpose of B is taken. The resulting object is a list of lists with entries in $\mathbb{R}_{\min}$.

Parameters:

A - A matrix with semiring elements inputted as a list of lists.

B - A matrix with semiring elements inputted as a list of lists

Example:

```
minplus_list_multiplication([[1, 'id', 4], ['id', 2, 1]],
                            [[4, 7], [2, 'id'], [8, 'id']])


[[5, 8], [4, 'id']]
```

`A_to_the_k(A, k)`

Description: Returns the matrix A raised to the power of k. The function begins with a conditional that makes sure the matrix is square. Then, we assign i to the integer 1 and assign A_i to the input value A. Within a while loop, the function performs min-plus matrix multiplication on A and A_i, and then assigns i to i+1. The loop repeats until it runs with i:=k.

Parameters:

A - A square matrix with semiring elements inputted as a list of lists

k - A positive integer

Example:

```
A_to_the_k([[8, 0, 2, 1], [7, 3, 3, 6], [4, 5, 5, 6], [3, 3, 7, 9]], 4)
```

```
[[8, 7, 7, 8], [11, 10, 10, 11], [11, 8, 10, 9], [10, 7, 9, 8]]
```

`A_list_power_list(A, k):`

Description: This function is designed solely for experimentation. It does not actually return any value, but rather prints a list of positive integer powers of a min-plus matrix A. This can be especially useful in studying the Power Algorithm. First, the matrix A is printed. Then, the variable i is assigned to the integer 2; a while loop prints powers of A and assigns i to the next greatest integer value until i:=k.

Parameters:

A - a square matrix with semiring elements inputted as a list of lists.

k - a positive integer.

Example:

```
A_list_power_list([[8, 0, 2, 1], [7, 3, 3, 6], [4, 5, 5, 6], [3, 3, 7, 9]], 8)


1: [[8, 0, 2, 1], [7, 3, 3, 6], [4, 5, 5, 6], [3, 3, 7, 9]]
2: [[4, 3, 3, 6], [7, 6, 6, 8], [9, 4, 6, 5], [10, 3, 5, 4]]
3: [[7, 4, 6, 5], [10, 7, 9, 8], [8, 7, 7, 10], [7, 6, 6, 9]]
4: [[8, 7, 7, 8], [11, 10, 10, 11], [11, 8, 10, 9], [10, 7, 9, 8]]
5: [[11, 8, 10, 9], [14, 11, 13, 12], [12, 11, 11, 12], [11, 10, 10, 11]]
6: [[12, 11, 11, 12], [15, 14, 14, 15], [15, 12, 14, 13], [14, 11, 13, 12]]
7: [[15, 12, 14, 13], [18, 15, 17, 16], [16, 15, 15, 16], [15, 14, 14, 15]]
8: [[16, 15, 15, 16], [19, 18, 18, 19], [19, 16, 18, 17], [18, 15, 17, 16]]
```

`A_plus(A)`

Description: Returns the A^+ matrix described in Section 4, Chapter 1. It includes a conditional to make the the input is a square matrix. Then the function appends k iterations of the A_to_the_k() function to a list. Using a while loop, the function then performs min-plus matrix addition over the list of matrix powers.

Parameters:

A - A square matrix with semiring elements inputted as a list of lists.

Example:

```
A_plus([[4, 3, 1], [2, 4, 0], [1, 'id', 3]])


[[2, 3, 1], [1, 4, 0], [1, 4, 2]]
```

`Karps_Algorithm(A, j)`

Description: Returns the max-plus eigenvalue of an irreducible matrix. A need not be irreducible for the algorithm to run. In fact, we designed the function with this in mind, knowing that the crux of our analysis would be seeing where, how, and why spectral methods of irreducible systems break down in the reducible case. Moreover, what is perhaps unique about this version of the function is that it gives a print-out of the various steps of the algorithm. This feature was included to aide the aforementioned analysis. The first loop constructs the appropriate canonical vector, given the user's choice of j. Another for loop proceeds to build a list of the $\mathbf{x}(k)$ vectors using min-plus matrix multiplication. A list of these vectors is printed out, for the user's convenience. Then, step (3) of Karp's Algorithm is performed, with the various difference quotients being printed along the way. Special consideration had to be given to 'id' here, as our tropical_multiplication() function does not take into account the fact that $\epsilon - \epsilon = 0$. The maximum is taken among the difference quotients, and the minimum of these maximums is returned as the function's output.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

j - An integer representing the starting node for the algorithm

Example:

```
Karps_Algorithm([[4, 3, 4], [2, 4, 3], [1, 3, 3]], 3)

[[['id'], ['id'], [0]], [[4], [3], [3]], [[6], [6], [5]], [[9], [8], [7]]]

( 9 - id ) / 3
( 9 - 4 ) / 2
( 9 - 6 ) / 1


( 8 - id ) / 3
( 8 - 3 ) / 2
( 8 - 6 ) / 1


( 7 - 0 ) / 3
( 7 - 3 ) / 2
( 7 - 5 ) / 1


7/3
```

`Eigenspace(A, lmda)`

Description: Returns a min-plus eigenvector of the matrix A associated with the eigenvalue lmda. Note that we use lmda because lambda is recognized by Python as an expression form. The process employed by this function is, essentially, the algorithm given in Section 1, Chapter 5 of [**3**] for computing the tropical eigenspace of a given eigenvector. First, a for loop computes the A_λ matrix by performing min-plus multiplication with each element of A and -lmda. We then use our A_plus() function to calculate A_λ^+, and a for loop looks for columns in the transpose of A_λ^+, for which the diagonal entry is 0. These columns form the eigenspace associated with lmbda, and they are printed out before the function returns a single eigenvector as output.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

lmda - An eigenvalue of A

Example:

```
Eigenspace([[4, 3, 4], [2, 4, 3], [1, 3, 3]], 7/3)


[[[0], [2/3], [4/3]], [[-2/3], [0], [2/3]], [[-4/3], [-2/3], [0]]]
[[0], [2/3], [4/3]]
```

Max-Plus Package

```
maxplus_addition(a, *b)
```

Description: This function is part of the foundation of our package. It computes the max-plus sum of any finite number of elements in $\mathbb{R}_{\max}$. The parameter *b is stored as a tuple which we then convert to a list so that we have a mutable Python object. The function goes through a series of conditionals to determine which parameters, if any, are our additive identity. If a == 'id', then it is ignored. If *b contains the additive identity, then the list.pop() method is used to eliminate all such occurrences. Eventually, we consider only finite entries, over which we use Python's built-in max() function. Of course, if all entries are infinite, then the function returns 'id'.

Parameters:

a - A semiring element

*b - Any number of semiring elements

Example:

```
maxplus_addition(17, 'id')


17
```

```
maxplus_addition(17, 'id', 4, 'id')

17
```

`tropical_multiplication(a, *b)`

Description: The multiplication function forms the other half of our package's foundation. As with addition, this function receives any finite number of $\mathbb{R}_{\min}$ elements. Conditionals search for the additive identity, and if it is at all present, our output is 'id'. Otherwise, Python's built-in sum() function "multiplies" finite entries.

Parameters:

a - A semiring element

*b - Any number of semiring elements

Example:

```
tropical_multiplication(7,'id',2)

'id'

tropical_multiplication(7,2)

9
```

`maxplus_list_addition(A, B)`

Description: Returns the max-plus sum of two matrices. The function begins with two conditionals that make sure our matrix inputs are of equal dimension. Using Python's len() function, we test the length of the outer list to make sure each matrix has the same number of rows. Then, we verify that the length of the first list entries within the outer lists match in length. In this regard, the function is limited in that it only makes certain that the

dimensions of the first columns of each matrix match. Using nested for loops, the function iterates over each matrix row, constructing an empty list for each row of input A. The next for loop iterates over each element within the matrix rows and appends the max-plus sum of corresponding elements to the aforementioned empty list. These altered lists then become the rows of the output matrix.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

B - A matrix with semiring elements inputted as a list of lists

Example:

```
maxplus_list_addition([[23, 'id', 4], [-4, 5, -6], [2, 3, -1]],
                      [[6, 7, 3], [1, 10, 'id'], [1, 8, 6]])


[[23, 7, 4], [1, 10, -6], [2, 8, 6]]
```

`maxplus_dot_product(x, y)`

Description: This function returns the min-plus dot product of two vectors. First, a conditional checks to makes sure the vectors are of equal dimension. Next, a for loop iterates over a list of integers equal in length to the input vectors to perform max-plus multiplication on corresponding elements. After each max-plus product is appended to a list called "dot_product", a for loop takes the max-plus sum of the "dot_product" list.

Parameters:

x - A vector with semiring elements

y - A vector with semiring elements

Example:

```
maxplus_dot_product([13, 8, 4], [14, 16, 'id'])
```

```
27
```

`list_transpose(A)`

Description: This function plays a crucial role in our matrix multiplication functions. A for loop establishes empty lists that will be the new rows of $A^{\intercal}$. A second for loop appends the element A[i][j] to the position A[j][i] in our new lists of lists.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

Example:

```
list_transpose([[1, 2, 3], ['id', 3, 5]])


[[1, 'id'], [2, 3], [3, 5]]
```

`maxplus_list_multiplication(A, B)`

Description: Returns the max-plus product of two matrices. A conditional is given to make sure that the number of columns in the first entry equals the number of rows in the second entry. A second conditional performs multiplication for the special case where B is a matrix with one column. Refining our package would include altering this function so that it could accept vector entries, rather than treating one-column matrices and vectors as different objects. In both cases, where len(B) == 1 and len(B)>1, the function proceeds to take the max-plus dot product of the rows of A with the column(s) of B and appends these products to lists that are then appended to yet another list. Note that for the third conditional, when B has more than one column, the transpose of B is taken. The resulting object is a list of lists with entries in $\mathbb{R}_{\max}$.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

B - A matrix with semiring elements inputted as a list of lists

Example:

```
maxplus_list_multiplication([[1, 'id', 4], ['id', 2, 1]],
                            [[4, 7], [2, 'id'], [8, 'id']])
```

```
[[12, 8], [9, 'id']]
```

`A_to_the_k(A, k)`

Description: Returns the matrix A raised to the power of k. The function begins with a conditional that makes sure the matrix is square. Then, we assign i to the integer 1 and assign A_i to the input value A. Within a while loop, the function performs max-plus matrix multiplication on A and A_i, and then assigns i to i+1. The loop repeats until it runs with i:=k.

Parameters:

A - A square matrix with semiring elements inputted as a list of lists

k - A positive integer

Example:

```
A_to_the_k([[8, 0, 2, 1], [7, 3, 3, 6], [4, 5, 5, 6], [3, 3, 7, 9]], 4)
```

```
[[32, 24, 26, 28], [31, 27, 31, 33], [28, 27, 31, 33], [30, 30, 34, 36]]
```

`A_list_powers(A, k):`

Description: This function is designed solely for experimentation. It does not actually return any value, but rather prints a list of positive integer powers of a max-plus matrix A. This can be especially useful in studying the Power Algorithm. First, the matrix A is printed. Then, the variable i is assigned to the integer 2; a while loop prints powers of A and assigns i to the next greatest integer value until i:=k.

Parameters:

A - a square matrix with semiring elements inputted as a list of lists.

k - a positive integer.

Example:

```
A_list_powers([[8, 0, 2, 1], [7, 3, 3, 6], [4, 5, 5, 6], [3, 3, 7, 9]], 8)

1 : [[8, 0, 2, 1], [7, 3, 3, 6], [4, 5, 5, 6], [3, 3, 7, 9]]

2 : [[16, 8, 10, 10], [15, 9, 13, 15], [12, 10, 13, 15], [12, 12, 16, 18]]

3 : [[24, 16, 18, 19], [23, 18, 22, 24], [20, 18, 22, 24], [21, 21, 25, 27]]

4 : [[32, 24, 26, 28], [31, 27, 31, 33], [28, 27, 31, 33], [30, 30, 34, 36]]

5 : [[40, 32, 35, 37], [39, 36, 40, 42], [36, 36, 40, 42], [39, 39, 43, 45]]

6 : [[48, 40, 44, 46], [47, 45, 49, 51], [45, 45, 49, 51], [48, 48, 52, 54]]

7 : [[56, 49, 53, 55], [55, 54, 58, 60], [54, 54, 58, 60], [57, 57, 61, 63]]

8 : [[64, 58, 62, 64], [63, 63, 67, 69], [63, 63, 67, 69], [66, 66, 70, 72]]
```

`A_plus(A)`

Description: Returns the A^+ matrix described in Section 4, Chapter 1. It includes a conditional to make the the input is a square matrix. Then the function appends k iterations of the A_to_the_k() function to a list. Using a while loop, the function then performs max-plus matrix addition over the list of matrix powers.

Parameters:

A - a matrix with semiring elements inputted as a list of lists.

Example:

```
A_plus([[4, 3, 1], [2, 4, 0], [1, 'id', 3]])

[[12, 11, 9], [10, 12, 8], [9, 8, 9]]
```

`Karps_Algorithm(A, j)`

Description: Returns the max-plus eigenvalue of an irreducible matrix. A need not be irreducible for the algorithm to run. In fact, we designed the function with this in mind, knowing that the crux of our analysis would be seeing where, how, and why spectral methods of irreducible systems break down in the reducible case. Moreover, what is perhaps unique about this version of the function is that it gives a print-out of the various steps of the algorithm. This feature was included to aide the aforementioned analysis. The first loop constructs the appropriate canonical vector, given the user's choice of j. Another for loop proceeds to build a list of the $\mathbf{x}(k)$ vectors using max-plus matrix multiplication. A list of these vectors is printed out, for the user's convenience. Then, step (3) of Karp's Algorithm is performed, with the various difference quotients being printed along the way. Special consideration had to be given to 'id' here, as our tropical_multiplication() function does not take into account the fact that $\epsilon - \epsilon = 0$. The minimum is taken among the difference quotients, and the maximum of these minimums is returned as the function's output.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

j - An integer representing the starting node for the algorithm

Example:

```
Karps_Algorithm([[4, 3, 4], [2, 4, 3], [1, 3, 3]], 3)

( 12 - id ) / 3
```

```
( 12 - 4 ) / 2
( 12 - 8 ) / 1

( 11 - id ) / 3
( 11 - 3 ) / 2
( 11 - 7 ) / 1

( 10 - 0 ) / 3
( 10 - 3 ) / 2
( 10 - 6 ) / 1

 4
```

`Eigenspace(A, lmda)`

Description: Returns a max-plus eigenvector of the matrix A associated with the eigenvalue lmda. Note that we use lmda because lambda is recognized by Python as an expression form. The process employed by this function is, essentially, the algorithm given in Section 1, Chapter 5 of [**3**] for computing the eigenspace of a given eigenvector. First, a for loop computes the A_λ matrix by performing max-plus multiplication with each element of A and -lmda. We then use our A_plus() function to calculate A_λ^+, and a for loop looks for columns in the transpose of A_λ^+, for which the diagonal entry is 0. These columns form the eigenspace associated with lmbda, and they are printed out before the function returns a single eigenvector as output.

Parameters:

A - A matrix with semiring elements inputted as a list of lists

lmda - An eigenvalue of A

Example:

```
Eigenspace([[4, 3, 4], [2, 4, 3], [1, 3, 3]], 4)

[[[0], [-1], [0]], [[-2], [0], [-1]]]
[[0], [-1], [0]]
```

Appendix B: Sage Min-plus and Max-plus Functions

Min-Plus Program Package

Below is a printout of the code that comprises the packages described in the above Appendix.

```
def minplus_addition(a, *b):
    blist = list(b)
    if a == 'id' and 'id' in blist:
        for i in blist:
            if i == 'id':
                blist.remove('id')
        if not blist:
            return 'id'
        else:
            return min(blist)
    elif a == 'id' and 'id' not in blist:
        return min(blist)
    elif a != 'id' and 'id' in blist:
        for j in blist:
            if j == 'id':
                blist.remove('id')
        if not blist:
            return a
```

```
        else:
            return min(a, min(blist))
    else:
        return min(a, min(blist))
```

```
def tropical_multiplication(a, *b):
    if a == 'id':
        return 'id'
    elif 'id' in b:
        return 'id'
    else:
        return sum(b, a)
```

```
def minplus_list_addition(A, B):
    if len(A) != len(B):
        print "Error: Your matrices must be of equal dimension."
    elif len(A[0]) != len(B[0]):
        print "Error: Your matrices must be of equal dimension."
    else:
        sumlist = []
        for i in range(len(A)):
            c_i = []
            for j in range(len(A[0])):
                c_i.append(minplus_addition(A[i][j], B[i][j]))
            sumlist.append(c_i)
        return sumlist
```

```
def minplus_dot_product(x, y):
```

```
    if len(x) != len(y):
        print "Error: The vectors you enter must have the.."
                "...same number of entries."
    else:
        xdy=[]
        for i in range(len(x)):
            xdy_i = tropical_multiplication(x[i], y[i])
            xdy.append(xdy_i)
        dot_product = xdy[0]
        for j in range(1, len(xdy)):
            dot_product = minplus_addition(dot_product, xdy[j])
        return dot_product


def list_transpose(A):
    A_transpose = []
    for i in range(len(A[0])):
        At_i = []
        for j in range(len(A)):
            At_i.append(A[j][i])
        A_transpose.append(At_i)
    return A_transpose


def minplus_list_multiplication(A, B):
    if len(A[0]) != len(B):
        print "Error: The number of columns in your first entry..."
                "...must equal the number of rows in your second."
    elif len(B) == 1:
        prodlist = []
```

```
        c_list = []
        for k in range(len(A)):
             c_list.append(minplus_dot_product(A[k], B[0]))
        prodlist.append(c_list)
        return prodlist
    else:
        Bt = list_transpose(B)
        prodlist = []
        for i in range(len(A)):
            c_i = []
            for j in range(len(B[0])):
                c_i.append(minplus_dot_product(A[i], Bt[j]))
            prodlist.append(c_i)
        return prodlist
```

```
def A_to_the_k(A, k):
    i = 1
    A_i = A
    while i < k:
        A_i = minplus_list_multiplication(A, A_i)
        i += 1
    return A_i
```

```
def A_list_powers(A, k):
    print A
    i = 1
    A_i = A
    while i < k:
```

```
        A_i = minplus_list_multiplication(A, A_i)
        print A_i
        i += 1


def A_plus(A):
    if len(A) != len(A[0]):
        print "Error: This function only accepts square matrices."
    else:
        A_list = []
        for i in range(1, (len(A)+1)):
            A_i = A_to_the_k(A, i)
            A_list.append(A_i)
        j = 1
        aplus = A_list[0]
        while j <= ((len(A)-1)):
            aplus = minplus_list_addition(A_list[j], aplus)
            j += 1
        return aplus


def Karps_Algorithm(A, j):
    n = len(A)
    x_0 = []
    catalogue = []
    for l in range(1, n+1):
        if l == j:
            x_0.append(0)
        else:
            x_0.append('id')
    catalogue.append(list_transpose([x_0]))
    for k in range(1, n+1):
```

```
        x_k = minplus_list_multiplication(A, catalogue[k-1])
        catalogue.append(x_k)
    print catalogue, '\n'
    candidates = []
    for i in range(n):
        m_i = []
        for k in range(n):
            print '(', catalogue[n][i][0], '-', catalogue[k][i][0], ')', '/', n-k
            if catalogue[n][i][0] != 'id' and catalogue[k][i][0] != 'id':
                m_i.append((catalogue[n][i][0]-catalogue[k][i][0])/(n-k))
            elif catalogue[n][i][0] == 'id' and catalogue[k][i][0] == 'id':
                m_i.append(0)
        print '\n'
        if m_i:
            candidates.append(max(m_i))
    if candidates:
        eigvlu = min(candidates)
        return eigvlu
```

```
def Eigenspace(A, lmda):
    Alambda = []
    for i in range(len(A)):
        Alambda_i = []
        for j in range(len(A)):
            Alambda_i.append(tropical_multiplication(A[i][j], -lmda))
        Alambda.append(Alambda_i)
    Aplus = A_plus(Alambda)
    eigenspace = []
    for k in range(len(A)):
        if Aplus[k][k] == 0:
            a_k = list_transpose([Aplus[k]])
```

```
13              eigenspace.append(a_k)
14      print eigenspace
15      return eigenspace[0]
```

Max-Plus Program Package

```
1
2def maxplus_addition(a, *b):
3    blist = list(b)
4    if a == 'id' and 'id' in blist:
5        for i in blist:
6            if i == 'id':
7                blist.remove('id')
8        if blist == []:
9            return 'id'
10       else:
11           return max(blist)
12   elif a == 'id' and 'id' not in blist:
13       return max(blist)
14   elif a != 'id' and 'id' in blist:
15       for j in blist:
16           if j == 'id':
17               blist.remove('id')
18       if blist == []:
19           return a
20       else:
21           return max(a, max(blist))
22   else:
23       return max(a, max(blist))
```

```
1def tropical_multiplication(a, *b):
2    if a == 'id':
3        return 'id'
```

```
    elif 'id' in b:
        return 'id'
    else:
        return sum(b, a)


def maxplus_list_addition(A, B):
    if len(A) != len(B):
        print "Error: Your matrices must be of equal dimension."
    elif len(A[0]) != len(B[0]):
        print "Error: Your matrices must be of equal dimension."
    else:
        sumlist = []
        for i in range(len(A)):
            c_i = []
            for j in range(len(A[0])):
                c_i.append(maxplus_addition(A[i][j], B[i][j]))
            sumlist.append(c_i)
        return sumlist


def maxplus_dot_product(x, y):
    if len(x) != len(y):
        print "Error: The vectors you enter must have the.."
                "...same number of entries."
    else:
        xdy=[]
        for i in range(len(x)):
            xdy_i = tropical_multiplication(x[i], y[i])
```

```
            xdy.append(xdy_i)
        dot_product = xdy[0]
        for j in range(1, len(xdy)):
            dot_product = maxplus_addition(dot_product, xdy[j])
        return dot_product


def list_transpose(A):
    A_transpose = []
    for i in range(len(A[0])):
        At_i = []
        for j in range(len(A)):
            At_i.append(A[j][i])
        A_transpose.append(At_i)
    return A_transpose


def maxplus_list_multiplication(A, B):
    if len(A[0]) != len(B):
        print "Error: The number of columns in your first entry..."
                "...must equal the number of rows in your second."
    elif len(B) == 1:
        prodlist = []
        c_list = []
        for k in range(len(A)):
             c_list.append(maxplus_dot_product(A[k], B[0]))
        prodlist.append(c_list)
        return prodlist
    else:
        Bt = list_transpose(B)
```

```
        prodlist = []
        for i in range(len(A)):
            c_i = []
            for j in range(len(B[0])):
                c_i.append(maxplus_dot_product(A[i], Bt[j]))
            prodlist.append(c_i)
        return prodlist
```

```
def A_to_the_k(A, k):
    i = 1
    A_i = A
    while i < k:
        A_i = maxplus_list_multiplication(A, A_i)
        i += 1
    return A_i
```

```
def A_list_powers(A, k):
    print 1, ':', A, '\n'
    i = 2
    A_i = A
    while i <= k:
        A_i = maxplus_list_multiplication(A, A_i)
        print i, ':', A_i, '\n'
        i += 1
```

```

def A_plus(A):
    if len(A) != len(A[0]):
```

```
        print "Error: This function only accepts square matrices."
    else:
        A_list = []
        for i in range(1, (len(A)+1)):
            A_i = A_to_the_k(A, i)
            A_list.append(A_i)
        j = 1
        aplus = A_list[0]
        while j <= ((len(A)-1)):
            aplus = maxplus_list_addition(A_list[j], aplus)
            j += 1
        return aplus
```

```
def Karps_Algorithm(A, j):
    n = len(A)
    x_0 = []
    catalogue = []
    for l in range(1, n+1):
        if l == j:
            x_0.append(0)
        else:
            x_0.append('id')
    catalogue.append(list_transpose([x_0]))
    for k in range(1, n+1):
        x_k = maxplus_list_multiplication(A, catalogue[k-1])
        catalogue.append(x_k)
    candidates = []
    for i in range(n):
        m_i = []
        for k in range(n):
            print '(', catalogue[n][i][0], '-', catalogue[k][i][0], ')', '/', n-k
```

```
            if catalogue[n][i][0] != 'id' and catalogue[k][i][0] != 'id':
                m_i.append((catalogue[n][i][0]-catalogue[k][i][0])/(n-k))
            elif catalogue[n][i][0] == 'id' and catalogue[k][i][0] == 'id':
                m_i.append(0)
        print '\n'
        if m_i:
            candidates.append(min(m_i))
    if candidates:
        eigvlu = max(candidates)
        return eigvlu
```

```
def Eigenspace(A, lmda):
    Alambda = []
    for i in range(len(A)):
        Alambda_i = []
        for j in range(len(A)):
            Alambda_i.append(tropical_multiplication(A[i][j], -lmda))
        Alambda.append(Alambda_i)
    Aplus = A_plus(Alambda)
    eigenspace = []
    for k in range(len(A)):
        if Aplus[k][k] == 0:
            a_k = list_transpose([Aplus[k]])
            eigenspace.append(a_k)
    print eigenspace
    return eigenspace[0]
```

Bibliography

[1] Anne Spalding, *Min-Plus Algebra and Graph Domination*, PhD dissertation, University of Colorado at Denver, 1998.

[2] Bernd Heidergott, Geert Jan Olsder, and Jacob van der Woude, *Max Plus at Work: Modeling and Analysis of Synchronized Systems*, Princeton University Press, Princeton, 2006.

[3] Diane Maclagan and Bernd Sturmfels, *Introduction to Tropical Geometry*, University of Warwick, 2009.

[4] Francois Baccelli, Guy Cohen, Gert Jan Olsder, and Jean-Pierre Quadrat, *Synchronization and Linearity: An Algebra For Discrete Event Systems*, Wiley, Hoboken, 1992.

[5] R. Balakrishnan and K. Ranganathan, *A Textbook of Graph Theory*, Springer, New York, 2012.

[6] R. A. Cuninghame-Green, *Minimax Algebra*, Springer-Verlag, Berlin, 1979.

[7] R. A. Cuninghame-Green, "Describing Industrial Processes with Interference and Approximating Their Steady-State Behaviour," *OR* **13** (1962), 95–100.

[8] Richard M. Karp, "A Characterization of the Minimum Cycle Mean in a Digraph," *Discrete Mathematics* **23** (1978), 309–311.

[9] Robert Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing* **1** (1972), 146–160.

[10] Travis Scrimshaw, "Tropical Semirings," *Sage Mathematics Software*, 28 Apr. 2013.

[11] William A. Stein et al, *Sage Mathematics Software*, (Version 6.1.1), 2014.